2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

Satellite Maneuver Evaluation Tool

by

Carlos Ismael Noriega

September 1990

Thesis Advisors: Herschel H. Loomis, Jr.
Michael J. Zyda

Approved for public release; distribution is unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DCLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School — 6b. OFFICE SYMBOL (If Applicable) 39	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (city, state, and ZIP code) Monterey, CA 93943-5000	7b. ADDRESS (city, state, and ZIP code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 6b. OFFICE SYMBOL (If Applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (city, state, and ZIP code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Satellite Maneuver Evaluation Tool

12. PERSONAL AUTHOR(S)
Noriega, Carlos I.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (year, month,day)	15. PAGE COUNT
Master's Thesis	FROM TO	September 1990	138

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES

FIELD	GROUP	SUBGROUP

18. SUBJECT TERMS (continue on reverse if necessary and identify by block number)
Orbital mechanics, simulatio·ι, graphics workstation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

When first introduced to orbital mechanics, students often experience difficulty in visualizing a satellite's actual path through space. The Satellite Mar.euver Evaluation Tool (SMET) seeks to alleviate that problem. SMET is a three-dimensional color graphics simulation of satellites in flight. It allows a student to interactively modify a satellite's orbital parameters and see the effects as the satellites' positions are updated continuously with respect to time. The user can change parameters by defining a maneuver or by directly entering a change through keyboard or dial inputs. SMET offers the user the capability to demonstrate difficult concepts, as well as a method to simulate actual satellite maneuvers. Instructors can videotape SMET sessions for classroom demonstrations. Instantaneous images can also be saved for redisplay or for printout.

This thesis provides a background on the mathematical formulas modeled by SMET. It also includes a detailed user's guide for SMET.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Herschel H. Loomis, Jr.	22b. TELEPHONE (Include Area Code) (408) 646-9767 — 22c. OFFICE SYMBOL EC/LM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted — SECURITY CLASSIFICATION OF THIS PAGE

All other editions are obsolete — Unclassified

Approved for public release; distribution is unlimited.

SATELLITE MANEUVER EVALUATION TOOL

by

Carlos Ismael Noriega
Captain, United States Marine Corps
B.S., University of Southern California, 1981

Submitted in partial fulfillment
of the requirements for the degrees of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(SPACE SYSTEMS OPERATIONS)

and

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1990

Author: ______________________
Carlos Ismael Noriega

Approved by: ______________________
Herschel H. Loomis, Jr., Thesis Advisor

____
Michael J. Zyda, Thesis Advisor

____
Rudolph Panholzer, Chairman, Space Systems Academic Group

____
Robert B. McGhee, Chairman, Department of Computer Science

ABSTRACT

When first introduced to orbital mechanics, students often experience difficulty in visualizing a satellite's actual path through space. The Satellite Maneuver Evaluation Tool (SMET) seeks to alleviate that problem. SMET is a three-dimensional color graphics simulation of satellites in flight. It allows a student to interactively modify a satellite's orbital parameters and see the effects as the satellites' positions are updated continuously with respect to time. The user can change parameters by defining a maneuver or by directly entering a change through keyboard or dial inputs. SMET offers the user the capability to demonstrate difficult concepts, as well as a method to simulate actual satellite maneuvers. Instructors can videotape SMET sessions for classroom demonstrations. Instantaneous images can also be saved for redisplay or for printout.

This thesis provides a background on the mathematical formulas modeled by SMET. It also includes a detailed user's guide for SMET.

TABLE OF CONTENTS

LIST OF FIGURES

* Figures produced using SMET PostScript picture saving option.

ACKNOWLEDGEMENTS

I thank Dr. Herschel H. Loomis, Jr. and Dr. Michael J. Zyda for the encouragement and support they provided in the development SMET. After an initial setback in another area of research, they allowed me the freedom of action necessary to complete my work on a tight time schedule.

I greatly appreciate the assistance provided by Captain Michael K. Weiderhold in locating the required mathematical support for the maneuver model used by SMET.

Most importantly I thank my wife, Wendy, and my children, Shayna and Ryan, for their patience and support during the past twenty-five months at the Naval Postgraduate School. Without them my achievements would not have been possible.

I. INTRODUCTION

Students in the Space Systems Operations and Space Systems Engineering curricula at the Naval Postgraduate School are required to develop a thorough understanding of orbital mechanics. They use equations, tables of data and two-dimensional static illustrations to see the relationship between a satellite's orbital parameters and its path in space. These students, most of whom arrive with little or no background in the field, often have difficulty in visualizing the actual path of a satellite's flight. There is no readily available tool assist in learning orbital mechanics; a tool which would allow a student to interactively modify a satellite's orbital parameters and see the effects.

A mathematical relationship can be represented as two-dimensional graph. A graph conveys information more readily to human beings than does a table of the underlying numbers. But if more than two variables can be manipulated such representations quickly loose there utility. Adding a third dimension to a graph and varying it with time can be used to portray more information. But that format may not be intuitive to the user. Visual simulation is another approach to be considered. Visual simulation is the creation, by computer, of a realistic, easily modified, moving image from the mathematical model of a phenomenon. Realism implies high-resolution, color graphics. [Ref. 1:p. 3-5]

The Satellite Maneuver Evaluation Tool (SMET) is a three-dimensional color graphics simulation of satellites in flight. It offers the user three ways of observing a satellite's position with respect to the Earth. In one window, the user is placed in space at an arbitrary point of his choice. From this position, he sees the motion of the satellite around a rotating model of the Earth. Another

window allows the user to see that portion of the Earth which is visible to the satellite. A third window shows a Mercator projection map of the Earth with a plot of the satellite's ground track. After a user enters a satellite's parameters, the satellite's position is continuously updated with respect to time in all three windows.

Although SMET allows the user to change parameters through the use of dials or keyboard input, its greatest utility is in its ability to let the user execute a satellite maneuver at any given moment. A maneuver is entered as a velocity change (magnitude and direction) or as a combination of three maneuver components (flight path angle change, heading angle change and velocity magnitude change). The user can also enter the burnout parameters of a launch vehicle in order to place a satellite into its orbit.

The user is also able to read a satellite's position and velocity continuously in different coordinate systems and unit types. By allowing the user to enter as many satellites as he desires, SMET also lets him visualize the relationship between several satellites. As an example, the user could enter several identical satellites but vary one of the parameters to visually compare the effect of that variation. SMET also allows students to create illustrations and to program simulations.

SMET is written for any Silicon Graphics IRIS 4D/GT, GTX or VGX workstation. A personal computer implementation was considered and rejected for two reasons. One, there is no commonly available set of three-dimensional color graphics routines for personal computers that offers the functionality to generate the images desired for SMET. Second, graphics workstations are

becoming more affordable all the time. An IRIS is not an extravagant piece of hardware. Currently, there are at least 10 IRIS workstations around the campus.

The next three chapters provide a background on the mathematical equations used in SMET's simulation of satellite motion, maneuver and launch. The equations are not derived, in most cases, because that was considered outside of the scope of this document. Those derivations can be found in the references or other publications listed in the bibliography. All angles in trigonometric functions are in radians. Inverse trigonometric functions may require additional quadrant checking that is not shown. Although distances and velocities can be displayed in metric or english units, SMET actually stores values and performs all computations in canonical units. This, however, is transparent to the user. Chapter V provides detailed instructions on how to use SMET. Chapter VI discusses ways to modify the source code for specific simulations not currently modeled in SMET. Appendix A is a consolidated list of common variables in orbital mechanics. Appendix B has procedures for recording SMET sessions on video tape. It also provides directions on how to print out Postscript files generated by SMET. Appendix C is the source code listing for SMET.

II. SATELLITE MOTION

A. CLASSICAL ORBITAL PARAMETERS

A satellite's path in space can be described by five independent quantities. These five quantities, called "orbital elements" or parameters, are sufficient to completely describe the size, shape and orientation of the orbital ellipse. A sixth element is required to describe the satellite's position along the path at a particular point in time. [Ref. 2:p. 58]

There are several sets of variables used to describe a satellites position. SMET uses the classical set of six orbital elements. The semi-major axis (a) and eccentricity (e) describe the size and shape of the ellipse. The inclination (i), longitude of the ascending node (h) and argument of perigee (g) describe the orientation of the ellipse are depicted in Figure 2-1. The epoch (t_o) is the time of perigee passage. It is used as the reference from which to compute the satellite's position with respect to time.

B. OTHER STATIC PARAMETERS

Based on the orbital elements, several other parameters are computed to help describe the orbit. These elements do not change with respect to time. The following equations show how to compute the semi-minor axis (b), period (T), perigee range (r_p) and apogee range(r_a).

$$b = a\sqrt{1 - e^2} \qquad \text{[Ref. 2:p. 33]}$$

$$T = 2\pi\sqrt{\frac{a^3}{\mu}} \qquad \text{[Ref. 2:p. 33]}$$

$$r_p = a\,(1 - e) \qquad \text{[Ref. 2:p. 25]}$$

$$r_a = a\,(1 + e) \qquad \text{[Ref. 2:p. 25]}$$

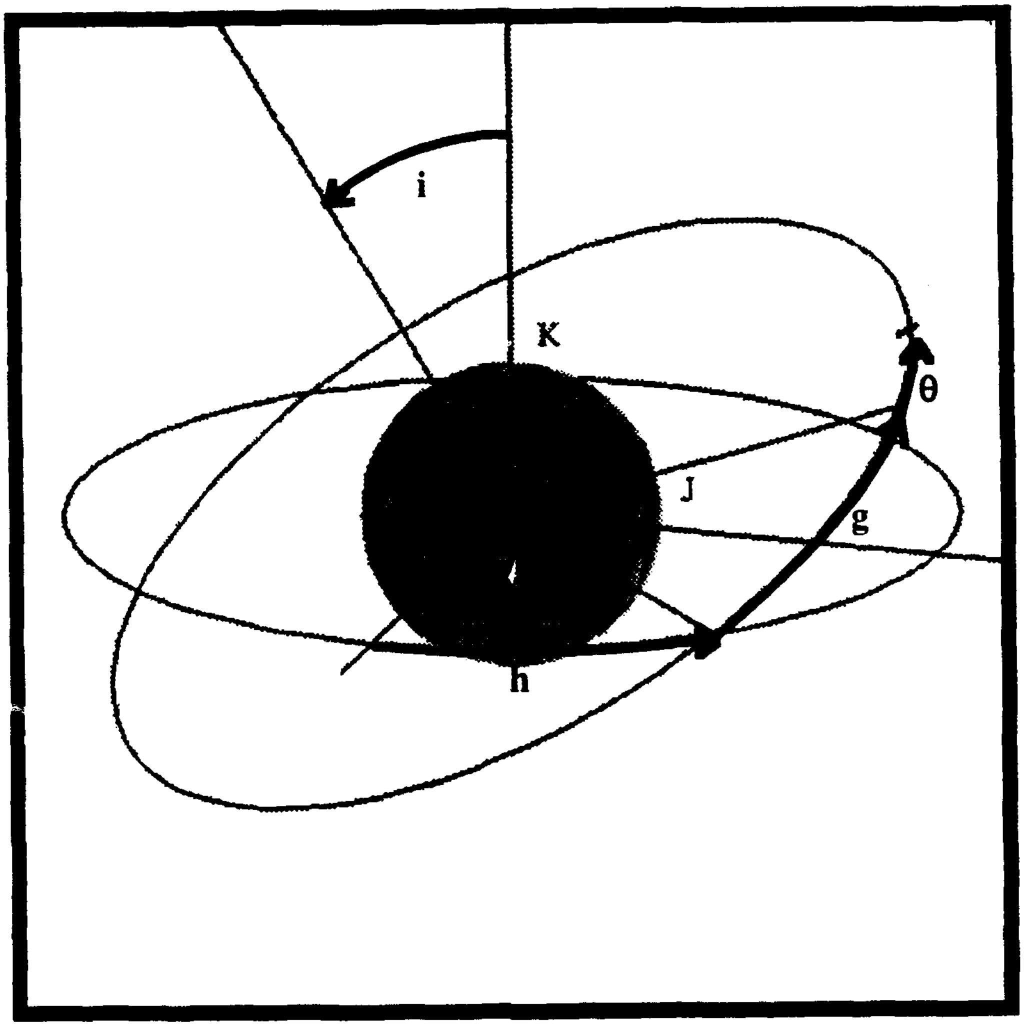

Figure 2-1. Orbital Elements

A transformation matrix $\tilde{\mathbf{R}}$ is also computed using the longitude of the ascending node (h), argument of perigee (g) and inclination (i). This matrix is used in drawing the satellite and its orbital ellipse. It is also used to convert perifocal coordinates into geocentric-equatorial coordinates.

$$\begin{bmatrix} \cos(h)\cos(g) - \sin(h)\sin(g)\cos(i) & -\cos(h)\sin(g) - \sin(h)\cos(g)\cos(i) & \sin(h)\sin(i) \\ \sin(h)\cos(g) + \cos(h)\sin(g)\cos(i) & -\sin(h)\sin(g) + \cos(h)\cos(g)\cos(i) & -\cos(h)\sin(i) \\ \sin(g)\sin(i) & \cos(g)\sin(i) & \cos(i) \end{bmatrix}$$

[Ref. 2:p. 83]

C. KEPLER PROBLEM

Given a satellite's position with respect to its classical elements, it is fairly simple to compute the elapsed time since epoch. This is described at the end of Chapter III. It is, however, more difficult to compute a position given the elapsed time. First the mean anomaly (M) must be computed using the following equation.

$$M = \sqrt{\frac{\mu}{a^3}}(t - t_o) - 2k\pi + M_o \qquad \text{[Ref. 2:p. 220]}$$

SMET defines epoch (t_o) at perigee. Since the mean anomaly and true anomaly are both zero at perigee, the last term (M_o) will drop out from the previous equation. The next step, computing the eccentric anomaly (E), is more difficult. To do this, the following equation must be solved for E. It is, in classical terms, referred to as the Kepler problem [Ref. 2:p. 220].

$$M = E - e \sin E \qquad \text{[Ref. 2:p. 220]}$$

This equation cannot be inverted in closed form to provide a function for E in terms of M. Therefore, another approach is required. The following Newton iteration scheme is used to solve for E.

$$E_0 = M$$

$$M_n = E_n - e \sin E_n \qquad \text{[Ref. 2:p. 221]}$$

$$E_{n+1} = E_n + \frac{M - M_n}{1 - e \cos E_n} \qquad \text{[Ref. 2:p. 222]}$$

The second and third equations are repeated until $M - M_n$ becomes sufficiently small. The true anomaly (θ) can be then computed from E.

$$\theta = \cos^{-1}\left(\frac{e - \cos E}{e \cos E - 1}\right) \qquad \text{[Ref. 2:p. 187]}$$

The true anomaly is depicted in Figure 2-1.

D. OTHER CHANGING PARAMETERS

Once the position of the satellite (i.e., true or eccentric anomaly) is computed, other descriptions of its position can be computed. First the range (r) of the satellite is computed using the eccentric anomaly:

$$r = a\,(1 - e \cos E) \qquad \text{[Ref. 3:p. C-7]}$$

Next, the coordinates in the perifocal coordinate system are computed:

$$P = r \cos\theta, \qquad \text{[Ref. 2:p. 72]}$$

$$Q = r \sin\theta, \qquad \text{[Ref. 2:p. 72]}$$

$W = 0$ by definition.

The coordinates in the geocentric-equatorial coordinate system (X, Y and Z) are then computed with the transformation matrix $\tilde{\mathbf{R}}$.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \tilde{\mathbf{R}} \begin{bmatrix} P \\ Q \\ W \end{bmatrix} \qquad \text{[Ref. 2:p. 83]}$$

The last set of coordinates to be computed are the latitude and longitude or the sub-satellite point. Using simple geometry, we get

$$\text{lat} = \sin^{-1}\left(\frac{Z}{r}\right),$$

$$\text{lon} = \tan^{-1}\left(\frac{Y}{X}\right).$$

These coordinates are fixed in space. Note that the earth rotates at an angular velocity of $\omega_{\oplus}$. In order to compute the longitude with respect to the rotated earth, subtract the earth's current rotation and allow for an arbitrary number of rotations to correct the previous equation. The corrected equation for longitude is

$$\text{lon} = \tan^{-1}\left(\frac{Y}{X}\right) - \omega_{\oplus} t + 2k\pi.$$

Although the magnitude of the velocity (v) is not a description of the position, it does provide some insight as to the motion of the satellite. It is also instrumental in computing orbital maneuvers. The following equation provides the instantaneous velocity of the satellite.

$$v = \sqrt{\mu\left(\frac{2}{r} - \frac{1}{a}\right)} \qquad \text{[Ref. 3:p. C-11]}$$

E. PERTURBATIONS

A perturbation is a variation in the path of the satellite not explained by the Keplerian model for two-body motion used up to this point. Perturbations result from the gravitational influence of other bodies, atmospheric drag and the oblate shape of the earth. The two principle effects are the regression of the line of nodes and the rotation of the line of apsides [Ref 2:p. 156]. These two effects change the longitude of the ascending node and argument of perigee, respectively. When turned on, SMET uses the following equations to model these two principle perturbations.

$$\dot{h} = \frac{-3\sqrt{\mu}\, J_2\, R_\oplus^2}{2\, a^{3.5}\,(1 - e^2)^2} \cos i \qquad \text{[Ref. 3:p. 4-4]}$$

$$\dot{g} = \frac{\sqrt{\mu}\, J_2\, R_\oplus^2}{a^{3.5}\,(1 - e^2)^2} \left(3 - \frac{15}{4}\sin^2 i\right) \qquad \text{[Ref. 3:p. 4-5]}$$

J_2 is the second order coefficient for perturbation accelerations due to the earth's nonsphericity. Other perturbations are more computationally intensive and provide changes to small to be visible with SMET. Therefore, SMET makes no attempt to model them.

F. FIELD OF VIEW CALCULATION

The Satellite View of SMET demonstrates how the altitude of a satellite effects the amount of the earth's surface visible to the satellite. This amount is referred to as the field of view (FOV) of the satellite. Note that the angle α in Figure 2-2 represents half of the FOV. If the range (r) is expressed in terms of earth radii (i.e., canonical units), the problem is simplified to:

$$\sin \alpha = \frac{1}{r}.$$

Thus

$$FOV = 2\sin^{-1}\left(\frac{1}{r}\right).$$

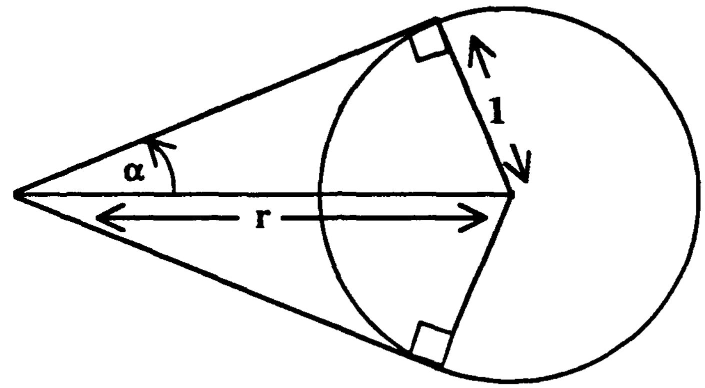

Figure 2-2. Field of View Calculation

III. SATELLITE LAUNCH

A. GENERAL DESCRIPTION

When a satellite is launched, it is normally attached to the top of a rocket. While the rocket is burning, the mass and velocity are constantly changing. Additionally, the effects of gravity and atmospheric drag change continuously. At the end of the rocket burn, the satellite will separate from the final stage of its rocket. The end effect of the rocket burn is to place the satellite at a particular point in space with a particular velocity. At this burnout point, the velocity vector of the satellite corresponds to a unique orbital path. SMET allows the user to enter parameters associatied this burnout point to start a satellite's motion. Modeling the complex interaction of the large number of variables that define the actual burn sequence is beyond the scope of SMET.

B. PARAMETERS

The values of the following variables at the time of burnout are used by SMET to compute the orbital parameters of the satellite. This is by no means the only way to define the burnout of the rocket. They were chosen because they are easy to visualize.

- v_o Velocity (with respect to a rotating earth)
- ϕ Latitude in earth-fixed coordinates
- λ Longitude in earth-fixed coordinates
- γ_o Flight Path Angle (measured upward from the local horizon plane)
- ψ_o Launch Azimuth (measured clockwise from North)
- alt Altitude above the surface of the earth

C. EARTH SPIN CORRECTION

The earth rotates about its axis in 23 hours and 56 minutes. This produces a surface velocity at the equator of approximately 465.10 m/sec (1524 ft/sec) [Ref. 2:p. 306]. The surface velocity is a function of latitude, decreasing as the poles are approached. This explains why eastward launches or equatorial launch points are more fuel efficient. SMET adds this induced velocity to the satellite velocity defined by the burnout parameters.

The velocity is broken up into components in a satellite fixed coordinate system. This is depicted in Figure 3-1. This system has a radial component, a North/South component and an East/West component. Using simple geometry, the following equations provide the solution for the velocity components.

$$v_R = v_o \sin \gamma_o$$

$$v_{N/S} = v_o \cos \gamma_o \cos \psi_o$$

$$v_{E/W_o} = v_o \cos \gamma_o \sin \psi_o$$

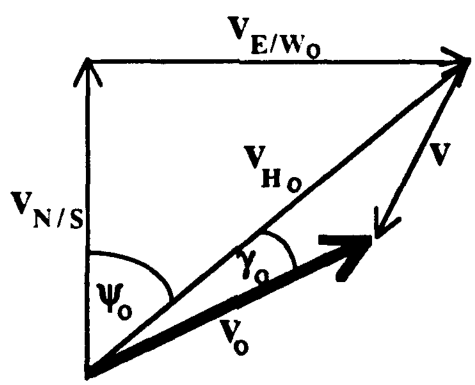

Figure 3-1. Original Velocity Vector

The earth rotation component (v_{SPIN}) is then computed from the latitude. $R_\oplus$ is the radius of the earth and $\omega_\oplus$ is the angular velocity of the earth.

$$v_{SPIN} = R_\oplus \omega_\oplus \cos\phi \qquad \text{[Ref. 2:p. 307]}$$

SMET adds v_{SPIN} to v_{E/W_o} and computes a new horizontal velocity component. The following equations, based on Figure 3-2, show the corrected values of velocity magnitude, flight path angle and azimuth.

$$v_{E/W} = v_{E/W_o} + v_{SPIN}$$

$$v_H = \sqrt{v_{N/S}^2 + v_{E/W}^2}$$

$$v = \sqrt{v_R^2 + v_H^2}$$

$$\gamma = \tan^{-1}\left(\frac{v_R}{v_H}\right)$$

$$\psi = \tan^{-1}\left(\frac{v_{E/W}}{v_{N/S}}\right)$$

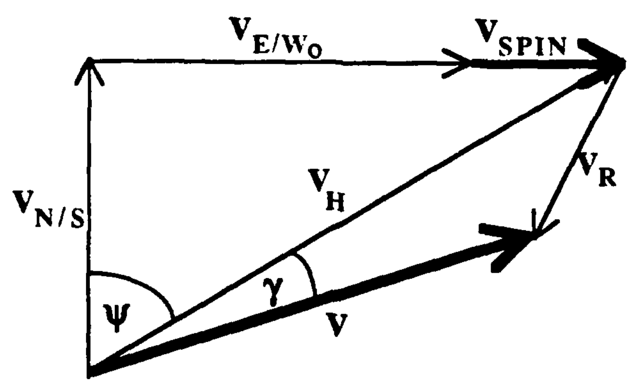

Figure 3-2. Corrected Velocity Vector

D. ORBIT PARAMETER CALCULATIONS

SMET uses the corrected burnout parameters to compute the orbital parameters. The inclination is a function of the azimuth and latitude.

$$i = \cos^{-1}\left(\sin\psi \cos\phi\right) \qquad \text{[Ref. 2:p. 142]}$$

The range is a function of the altitude.

$$r = 1 + \frac{alt}{R_\oplus}$$

The semi-major axis is a function of the range and velocity.

$$a = \frac{r}{2 - \frac{r\,v^2}{\mu}} \qquad \text{[Ref. 3:p. C-5]}$$

The semi-latus rectum is a function of the range and horizontal velocity.

$$p = \frac{r^2\,v_H{}^2}{\mu} \qquad \text{[Ref. 3:p. C-7]}$$

The eccentricity is a function of the semi-major axis and semi-latus rectum.

$$e = \sqrt{1 - \frac{p}{a}} \qquad \text{[Ref. 3:p. C-6]}$$

The true anomaly is a function of the range and semi-latus rectum.

$$\theta = \tan^{-1}\left(\frac{p\tan\gamma}{p - r}\right) \qquad \text{[Ref. 3:p. C-9]}$$

The arguments of latitude and longitude are functions of the latitude and azimuth.

$$u = \tan^{-1}\left(\frac{\sin\phi}{\cos\phi\cos\psi}\right) \qquad \text{[Ref. 3:p. A-7]}$$

$$\omega = \tan^{-1}\left(\frac{\sin\phi\sin\psi}{\cos\psi}\right) \qquad \text{[Ref. 3:p. A-7]}$$

The argument of perigee is a function of the argument of latitude and true anomaly.

$$g = u - \theta \qquad \text{[Ref. 3:p. C-9]}$$

To compute the longitude of the ascending node, note that $\lambda - \omega$ represents the distance between satellite's position in a rotational coordinate system and its position fixed coordinate system. Also note that $\omega_{\oplus}t$ represents the distance between an ascending node of 0° (fixed in space) and an ascending node that has rotated as much as the earth.

$$h = \lambda - \omega + \omega_{\oplus} t$$

As discussed in Chapter II, SMET computes the true anomaly as a function of the elapsed time since epoch (t_o). Therefore, after the true anomaly has been computed, the eccentric anomaly (E) and mean anomaly (M) are computed using the eccentricity (e) and true anomaly.

$$E = \cos^{-1}\left(\frac{e + \cos\theta}{1 + e\cos\theta}\right)$$

$$M = E - e \sin E$$

The last parameter, the epoch, is then computed using the mean anomoly, the semi-major axis (a) and current time (t).

$$t_o = t - \sqrt{\frac{a^3}{\mu}}M$$

IV. SATELLITE MANEUVERS

A. GENERAL DESCRIPTION

A satellite maneuver is executed by changing the velocity of the satellite. This change to the velocity vector can be in its magnitude, direction or both. A maneuver results in a new set of orbital parameters. A velocity change (Δv) is modeled as an instantaneous impulse. Therefore, the satellite's position or range does not change while the velocity does. The Δv vector can be broken up into component vectors (Δv_γ, Δv_ψ and Δv_α) which cause different effects simultaneously. These three vectors cause a flight path angle change, a heading angle change and a change in the magnitude of the velocity vector, respectively. Both Δv_γ and Δv_ψ change the direction of the velocity vector but do not change its magnitude. SMET computes the effects of each of the maneuvers independently and then updates the time of epoch. The time of epoch is computed such that the elapsed time will place the satellite in the same position based on the newly computed parameters.

B. FLIGHT PATH ANGLE CHANGE

A Δv_γ will cause a change in the flight path angle (γ). A flight path angle change is a maneuver in the orbital plane. The flight path angle is measured upward from the local horizon plane to the velocity vector [Ref. 3:p. 2-22]. The local horizon plane passes through the satellite and is perpendicular to the line passing through the center of the earth and the satellite. Δv_γ is chosen such that the magnitude of the velocity vector is unchanged (i.e., $v_1 = v_2$). These vectors

form an isosceles triangle as can be seen in Figure 4-1. Given $\Delta\gamma$, values for the eccentricity (e), argument of perigee (g), and true anomaly (θ) can be computed.

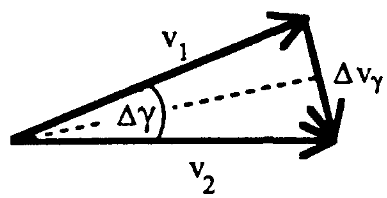

Figure 4-1. Flight Path Angle Change Vectors

The eccentricity changes as the impulse changes the shape of the orbit. The argument of perigee and true anomaly change as a result of the rotation of the line of apsides. This can be seen in Figure 4-2. Since Δv_γ has no out-of-plane component, there is also no change in the inclination (i) or the longitude of the ascending node (h). The following equation shows that the semi-major axis (a) does not change since the range and the magnitude of the velocity do not change.

$$a = \frac{r}{2 - \frac{r\,v^2}{\mu}} \qquad \text{[Ref. 3:p. C-5]}$$

The eccentricity can be computed given $\Delta\gamma$ by combining the following two equations for the cosine of the flight path:

$$\cos\gamma = \frac{\sqrt{\mu p}}{r v}, \qquad \text{[Ref. 3:p. 3-15]}$$

$$\cos\gamma_2 = \cos\gamma_1 (\cos\Delta\gamma - \tan\gamma_1 \sin\Delta\gamma). \qquad \text{[Ref. 3:p. 3-15]}$$

This results in

$$\frac{\sqrt{\mu p_2}}{r_2 v_2} = \frac{\sqrt{\mu p_1}}{r_1 v_1} (\cos\Delta\gamma - \tan\gamma_1 \sin\Delta\gamma).$$

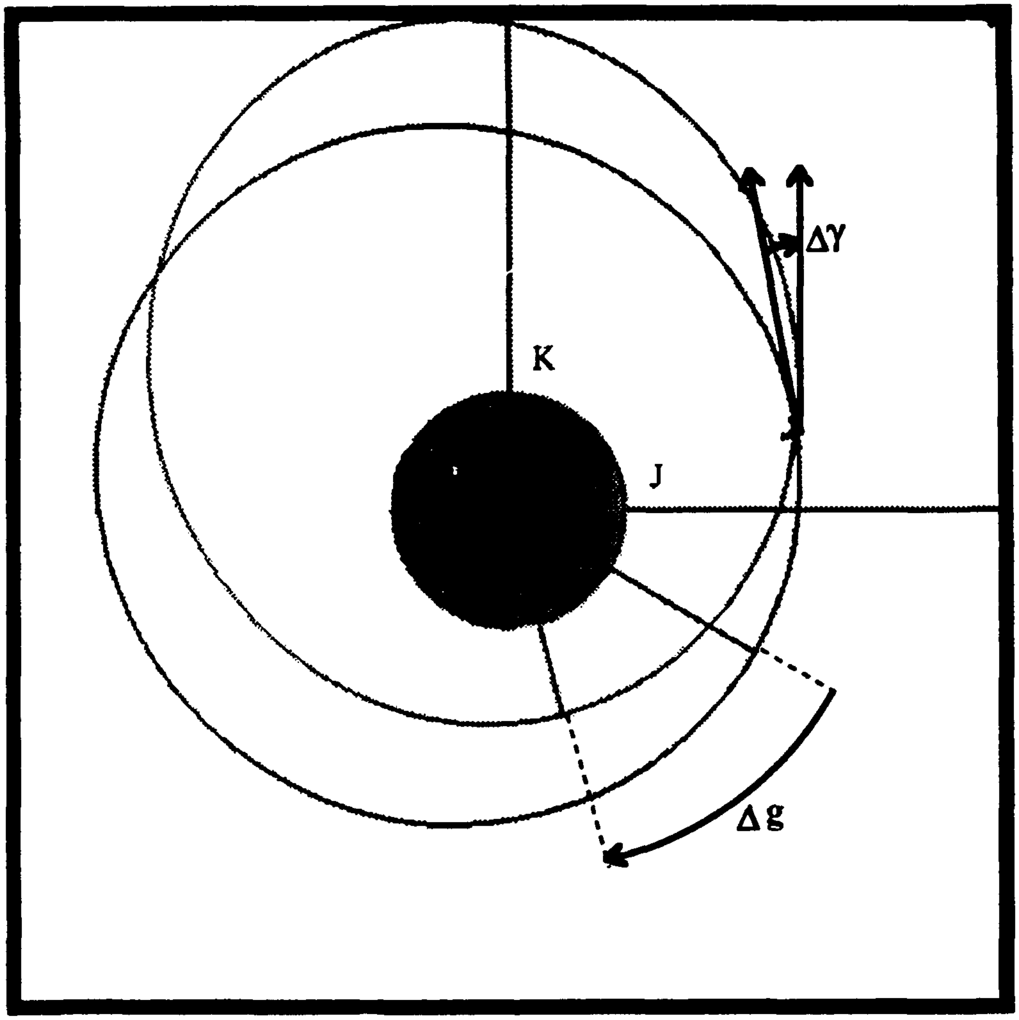

Figure 4-2. Flight Path Angle Change

Recalling that the range and magnitude of the velocity are unchanged, they can be removed from the previous equation.

$$\sqrt{\frac{p_2}{p_1}} = (\cos \Delta\gamma - \tan \gamma_1 \sin \Delta\gamma)$$

The semi-latus rectum is computed with

$$p = a\,(1 - e^2). \qquad \text{[Ref. 2:p. 24]}$$

Substituting this into the previous equation while recalling that the semi-major axis remains unchanged, results in

$$\sqrt{\frac{1 - e_2^2}{1 - e_1^2}} = (\cos \Delta\gamma - \tan \gamma_1 \sin \Delta\gamma),$$

$$1 - e_2^2 = (1 - e_1^2)(\cos \Delta\gamma - \tan \gamma_1 \sin \Delta\gamma)^2.$$

The tangent of the flight path angle can be computed from the eccentricity and eccentric anomaly.

$$\tan \gamma_1 = \frac{e_1 \sin E_1}{\sqrt{1 - e_1^2}} \qquad \text{[Ref. 3:p. C-10]}$$

Thus a new eccentricity can be computed with $\Delta\gamma$, e_1 and E_1.

$$e_2 = \sqrt{1 - (1 - e_1^2)\left(\cos \Delta\gamma - \left(\frac{e_1 \sin E_1 \sin \Delta\gamma}{\sqrt{1 - e_1^2}}\right)\right)^2}$$

Another equation relating the flight path angle to the satellite's position provides the means for computing the new true anomaly.

$$\theta = \tan^{-1}\left(\frac{p \tan \gamma}{p - r}\right) \qquad \text{[Ref. 3:p. C-9]}$$

Using $\tan \gamma_1$, which was computed earlier, we can compute the tangent of the new flight path angle.

$$\tan \gamma_2 = \frac{\tan \gamma_1 + \tan \Delta\gamma}{1 - \tan \gamma_1 \tan \Delta\gamma} \qquad \text{[Ref. 3:p. 3-19]}$$

Using $p_2 = a(1 - e_2^2)$, the new anomaly can thus be computed.

$$\theta_2 = \tan^{-1}\left(\frac{p_2 \tan \gamma_2}{p_2 - r}\right)$$

Since the satellite has not moved and the line of nodes has not changed, the argument of latitude (u) remains unchanged. Knowing that $g = u - \theta$, we can compute the new argument of perigee using the old argument of perigee and old true anomaly with the new true anomaly [Ref 3:p. 3-8].

$$g_2 = g_1 + \theta_1 - \theta_2$$

C. HEADING ANGLE CHANGE

A Δv_ψ will cause a change in the heading angle (ψ). A heading angle change is a maneuver out of the orbital plane. The heading angle is the angle in the local horizon plane measured from the line through the satellite and the north point and the velocity vector [Ref. 3:p. 2-22]. Again, Δv_ψ is chosen such that the magnitude of the velocity vector is unchanged (i.e., $v_1 = v_2$). These vectors form an isosceles triangle as can be seen in Figure 4-3. With $\Delta\psi$, values for the inclination (i), argument of perigee (g) and longitude of the ascending node (h) can be computed.

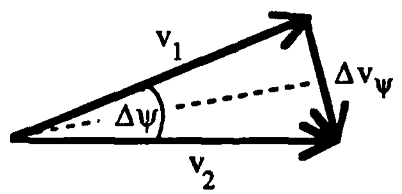

Figure 4-3. Heading Angle Change Vectors

The inclination changes as the orbital plane rotates about the line connecting the center of the earth and the current satellite position. This rotation can be seen in Figure 4-4. This rotation also changes the point where the orbital plane intersects with the equatorial plane, causing the ascending node to change. Finally, as the plane rotates, the distance between the ascending node and the current position as measured along the orbital path (the argument of latitude (u)) changes, causing the argument of perigee to change. Since $\Delta\psi$ has no in-plane component, there is no change to the shape or in-plane orientation of the orbit. Thus, the eccentricity (e), semi-major axis (a) and true anomaly (θ) remain unchanged.

Given $\Delta\psi$, the new inclination and argument of latitude can be computed using the following equations:

$$\cos i_2 = \cos i_1 \cos \Delta\psi + \sin i_1 \cos u_1 \sin \Delta\psi, \qquad \text{[Ref. 3:p. 3-6]}$$

$$\sin i_2 \cos u_2 = \sin i_1 \cos u_1 \cos \Delta\psi - \cos i_1 \sin \Delta\psi, \qquad \text{[Ref. 3:p. 3-6]}$$

$$u_1 = g_1 + \theta_1. \qquad \text{[Ref. 3:p. 3-8]}$$

The new inclination and argument of latitude are found by rearranging the previous equations.

$$i_2 = \cos^{-1}\left(\cos i_1 \cos \Delta\psi + \sin i_1 \cos(g_1+\theta_1) \sin \Delta\psi\right)$$

$$u_2 = \cos^{-1}\left(\frac{\sin i_1 \cos(g_1+\theta_1) \cos \Delta\psi - \cos i_1 \sin \Delta\psi}{\sin i_2}\right)$$

Since the true anomaly is unchanged (i.e., $\theta_1 = \theta_2$), the new argument of perigee is simply:

$$g_2 = u_2 - \theta_1.$$

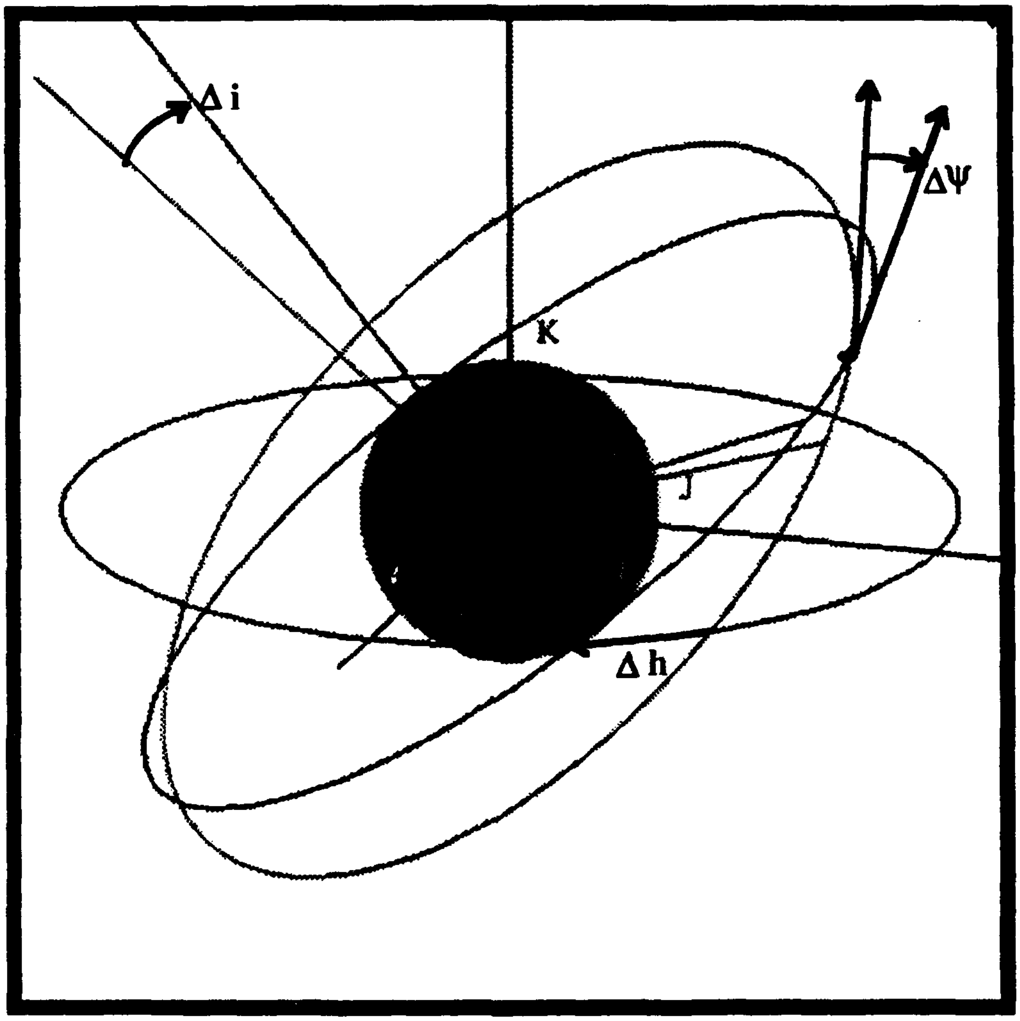

Figure 4-4. Heading Angle Change

The new longitude of the ascending node is computed by noting that the change in the longitude of the ascending node is equal to the change in the argument of longitude [Ref. 3:p. 3-11]. The argument of longitude is computed as follows:

$$\omega = \tan^{-1}\left(\frac{\sin u \cos i}{\cos u}\right) \qquad \text{[Ref. 3:p. A-8]}$$

$$\text{Thus } h_2 = h_1 + \tan^{-1}\left(\frac{\sin u_1 \cos i_1}{\cos u_1}\right) - \tan^{-1}\left(\frac{\sin u_2 \cos i_2}{\cos u_2}\right)$$

D. VELOCITY MAGNITUDE CHANGE

A Δv_α will change the magnitude of the velocity ($v_2 = v_1 + \Delta v_\alpha$) without changing the direction (Figure 4-5). Based on the new velocity magnitude, values for the semi-major axis (a) and eccentricity (e) can be computed. These changes are reflected as a change in the shape of the orbit (Figure 4-6). Since the heading angle and flight path angle are unchanged, the parameters defining the orientation of the orbital plane (i.e., i and h) are unchanged [Ref 2.:p. 3-20]. If the orbit was circular prior to the change, the argument of perigee (g) and true anomaly (θ) must also be set, since those two parameters have little meaning in a circular orbit. The velocity change at a position other than perigee or apogee will also cause a rotation of the line of apsides. Therefore, the argument of perigee and true anomaly must be computed as they were for a flight path angle change.

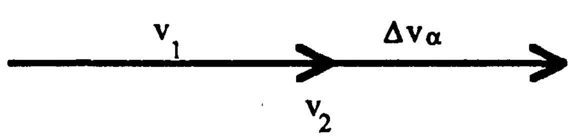

Figure 4-5. Velocity Magnitude Change Vectors

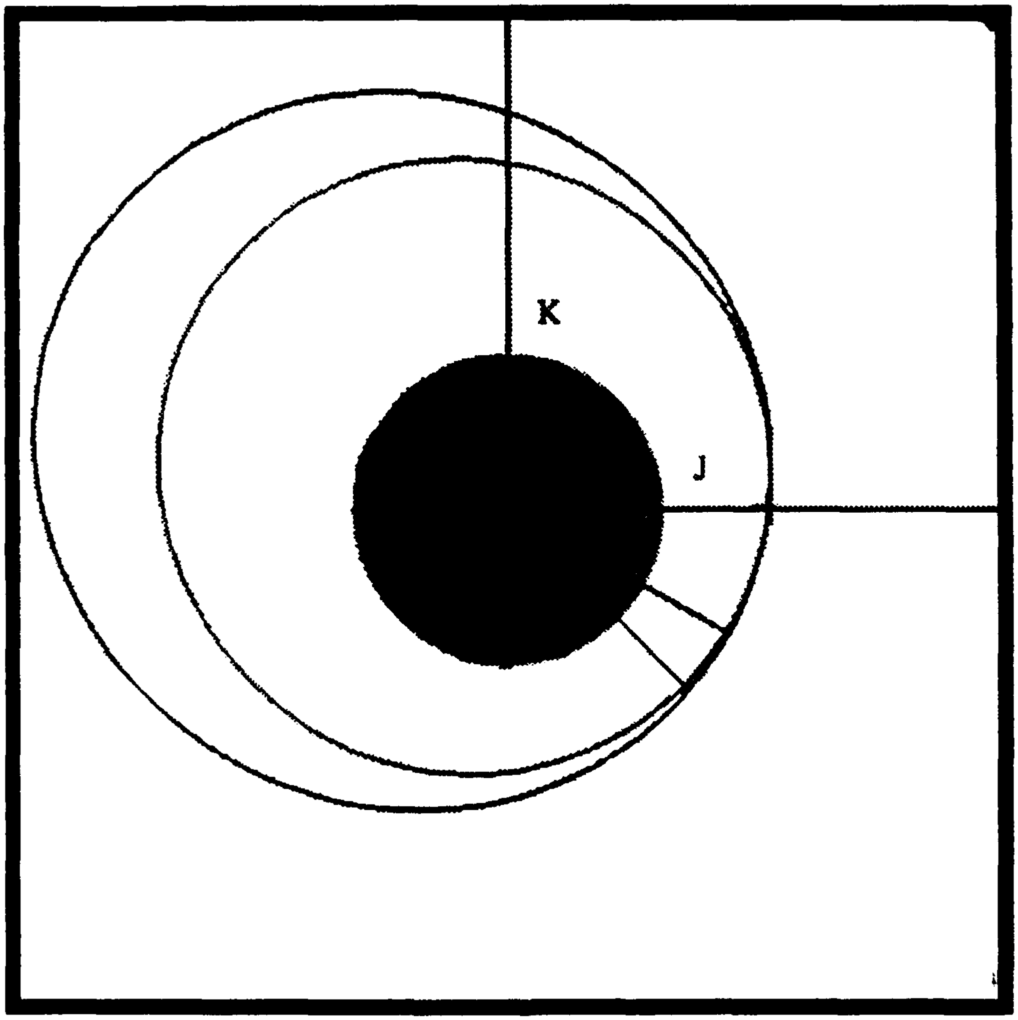

Figure 4-6. Velocity Magnitude Change

The semi-major axis and the eccentricity are computed with the following equations:

$$a_2 = \frac{r}{2 - \frac{r\,v_2^2}{\mu}}, \qquad \text{[Ref. 3:p. 3-20]}$$

$$e_2 = \sqrt{1 - \frac{r^2\,v_{H2}^2}{\mu\,a_2}} \quad \text{where } v_{H2} = v_2 \cos\gamma. \quad \text{[Ref. 3:p. 3-20]}$$

The flight path angle (γ) does not change since the velocity vector was not rotated. Therefore, the old values of e and E can be used to compute γ.

$$\tan\gamma = \frac{e_1 \sin E_1}{\sqrt{1 - e_1^2}} \qquad \text{[Ref. 3:p. C-10]}$$

If the original orbit was circular (i.e., $e_1 = 0$), the fact that the argument of latitude remains unchanged is used to compute a value for the argument of perigee and true anomaly. The current position will be perigee if there was an increase in velocity.

$$g_2 = u = g_1 + \theta_1$$

$$\theta_2 = u - g_2 = 0°$$

If the velocity decreased, the satellite will be at apogee of the new orbit.

$$g_2 = u - 180° = g_1 + \theta_1 - 180°$$

$$\theta_2 = u - g_2 = 180°$$

E. UPDATING EPOCH

Changes in the magnitude of the velocity and flight path angle cause a change to the true anomaly (θ). As in the satellite launch equations, the epoch is computed as a function of the true anomaly and the current time with respect to the eccentricity and semi-major axis of the new orbit.

F. IMPULSE VECTOR DIVISION

Given a velocity change vector (Δv) with an associated yaw and pitch, the in-plane and out-of-plane components (Δv_p and Δv_{op}) can be computed. Using Δv_p and Δv_{op}, the desired vector components (Δv_γ, Δv_ψ and Δv_α) can be computed. Unfortunately, Δv_γ, Δv_ψ and Δv_α are not orthogonal vector and, therefore, are not easily computed. Figure 4-7 depicts the relationship between the two sets of vectors. The same set of vectors is depicted again as part of Figure 4-10.

Earlier sections noted that Δv_γ and Δv_ψ formed isosceles triangles when added to the velocity vector. These isosceles triangles had associated angles ($\Delta\gamma$ and $\Delta\psi$) which were used to compute the flight path change maneuver or heading change maneuver. Similarly the new velocity (v_2), and not Δv_α, is actually used to compute the new orbital parameters for a velocity magnitude change maneuver. SMET, therefore, does not use Δv_γ, Δv_ψ and Δv_α. SMET actually computes $\Delta\gamma$, $\Delta\psi$ and v_2.

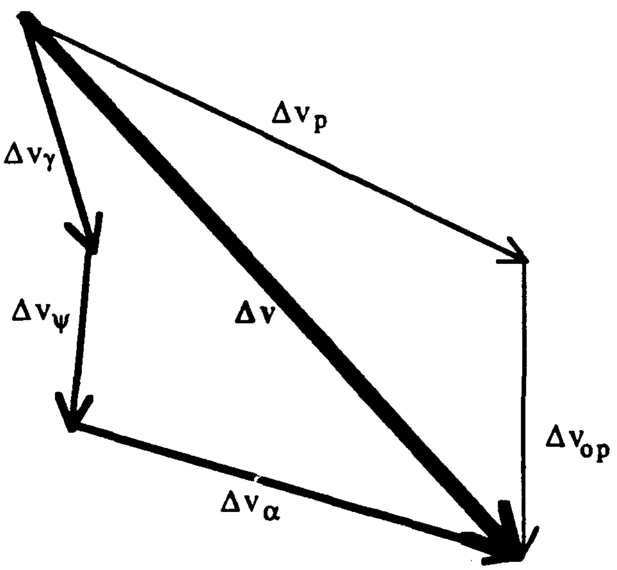

Figure 4-7. Division of Impulse Vector

The yaw associated with a maneuver is used to compute the in-plane and out-of-plane components of the Δv vector.

$$\Delta v_p = \Delta v \cos(Yaw)$$

$$\Delta v_{op} = \Delta v \sin(Yaw)$$

The pitch associated with the maneuver is used to compute that portion of the in-plane component (Δv_p) which is in the direction of motion and that portion which is perpendicular to the direction of motion. These vectors are added to the original velocity as depicted in Figure 4-8.

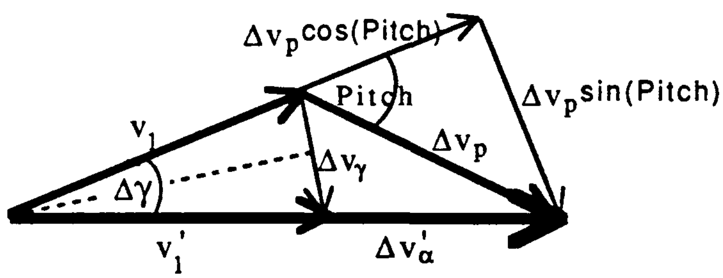

Figure 4-8. Computing Flight Path Angle Change

This right triangle provides the means with which to compute the change to flight path angle ($\Delta\gamma$).

$$\Delta\gamma = \tan^{-1}\left(\frac{\Delta v_p \sin(Pitch)}{v_1 + \Delta v_p \cos(Pitch)}\right)$$

The hypotenuse of the right triangle is computed for later use.

$$v_1' + \Delta v_\alpha' = \sqrt{(\Delta v_p \sin(Pitch))^2 + (v_1 + \Delta v_p \cos(Pitch))^2}$$

In order to compute the heading angle change ($\Delta\psi$), the out-of-plane components of Δv (Δv_{op}) are added to the previously computed in-plane components ($v_1' + \Delta v_\alpha'$). Figure 4-9 shows that a right triangle is formed again.

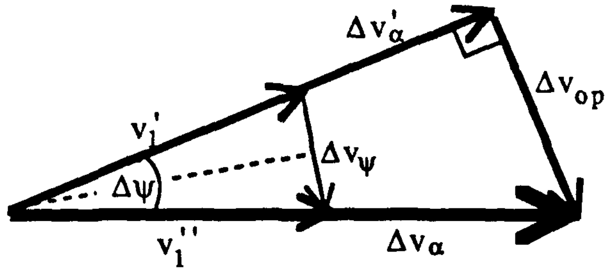

Figure 4-9. Computing Heading Angle Change

This right triangle provides the means with which to compute the change to the heading angle ($\Delta\psi$).

$$\Delta\psi = \tan^{-1}\left(\frac{\Delta v_{op}}{v'_1 + \Delta v'_\alpha}\right)$$

The hypotenuse of this right triangle ($v''_1 + \Delta v_\alpha$) is v_2. This can be seen in Figure 4-10 which combines the two previous figures.

$$v_2 = v''_1 + \Delta v_\alpha = \sqrt{\Delta v_{op}^2 + (v'_1 + \Delta v'_\alpha)^2}$$

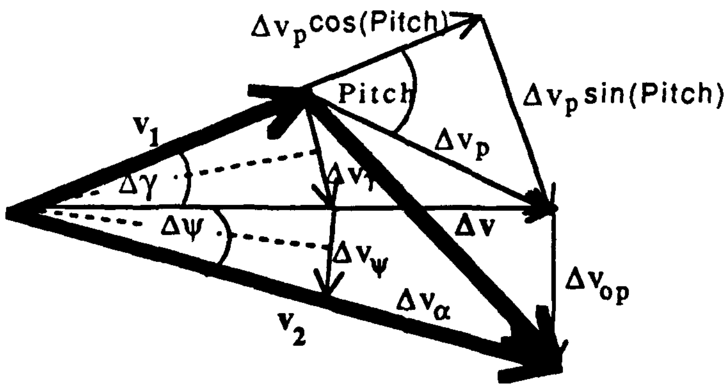

Figure 4-10. Computing New Velocity

V. USER'S GUIDE

A. OVERVIEW

This chapter describe SMET's display and user interface. Figure 5-1 shows the five windows used by SMET - the Main Window, two Auxiliary Windows, the Parameter Window and the Satellite List Window. The Main Window is a large square window at the center of the screen. There are two smaller Auxiliary Windows below the Main Window. These three windows are called viewing windows. The Parameter Window is a long narrow window on the left side of the screen. The Satellite List Window is a long narrow window on the right side of the screen.

SMET updates all satellite positions continuously with respect to time. One satellite is designated as the Current Satellite. This satellite is listed at the top of the Satellite List Window and has its parameters displayed in the Parameter Window. Additionally the satellite viewpoint, discussed later, belongs to the Current Satellite. All user inputs affect the Current Satellite. Any satellite can become the Current Satellite with the use of the Satellite List Window.

B. VIEWING WINDOWS

The three viewpoints discussed in this section can appear in any of the three viewing windows. These images are completely interchangeable. The user can switch an image from an Auxiliary Window into the Main Window by simply pressing the middle mouse button while the cursor is in the desired Auxiliary Window. The image that was in the Main Window will be moved to the selected Auxiliary Window.

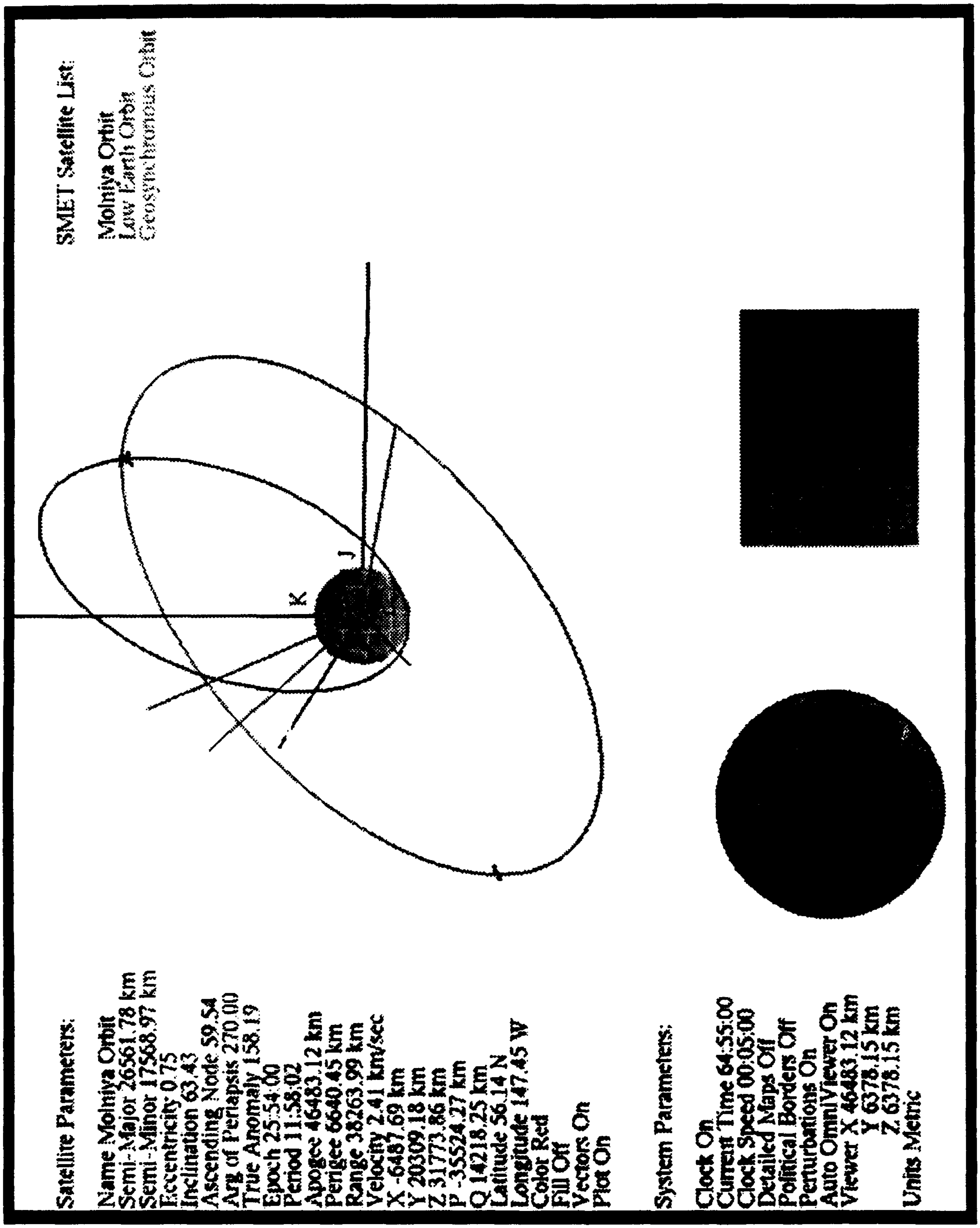

Figure 5-1. SMET Windows

1. OmniViewer View

The OmniViewer View allows the user to observe all satellites as they orbit around a rotating earth model as seen in Figure 5-2. Axes for the geocentric-equatorial coordinate system are displayed for user orientation. An orbital path is displayed for each satellite. The user can choose to draw it as a colored line or as a colored transparent ellipse. Normal and perigee vectors can also be displayed for any satellite to assist in user orientation.

The user's viewing position is referred to as the Viewer X, Y and Z in the Parameter Window. Changing the user's viewing position is discussed in SMET System Parameters. It can be set automatically or adjusted by the user. When Automatic OmniViewer is turned on, the viewer will be positioned far enough away to be able to see all orbital paths. The viewer position will be adjusted automatically whenever a satellite is added or any orbital path is changed as a result of a maneuver or user input. During satellite maneuvers or launches, Automatic OmniViewer can be turned off. This allows the user to see the event on a single scale.

2. Satellite View

The Satellite View allows the user to see that portion of the earth visible to the Current Satellite. The image is a "fish-eye lense" type of view. The user sees a circular image of the earth encircled by a ring colored in the Current Satellite's color. The center of the image corresponds with the Current Satellite's sub-satellite point. The outer edge represents the horizon as seen from the altitude of the Current Satellite. As the Current Satellite's altitude changes, the field of view is adjusted to maximize the size of the image. Therefore, the size of the image does not generally change. Figure 5-3 shows a typical view from a satellite in a Molniya orbit.

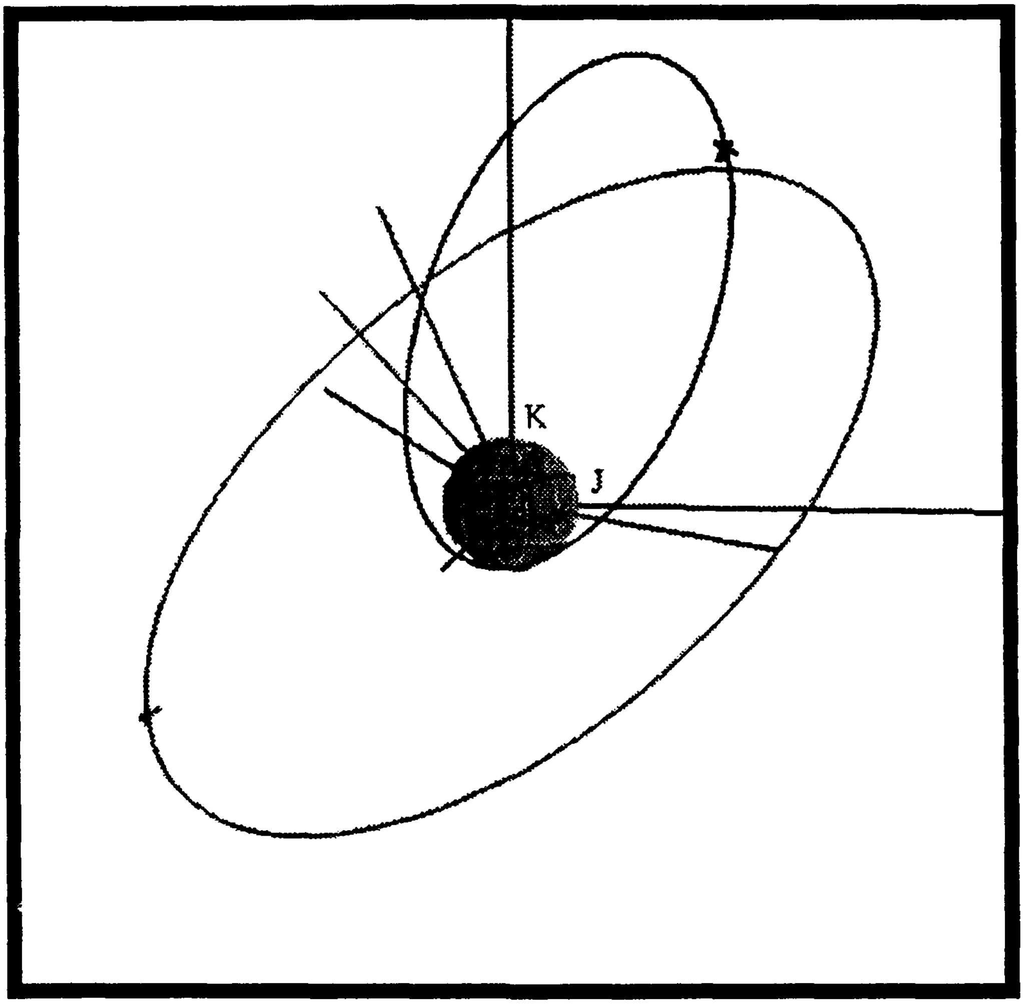

Figure 5-2. OmniViewer View

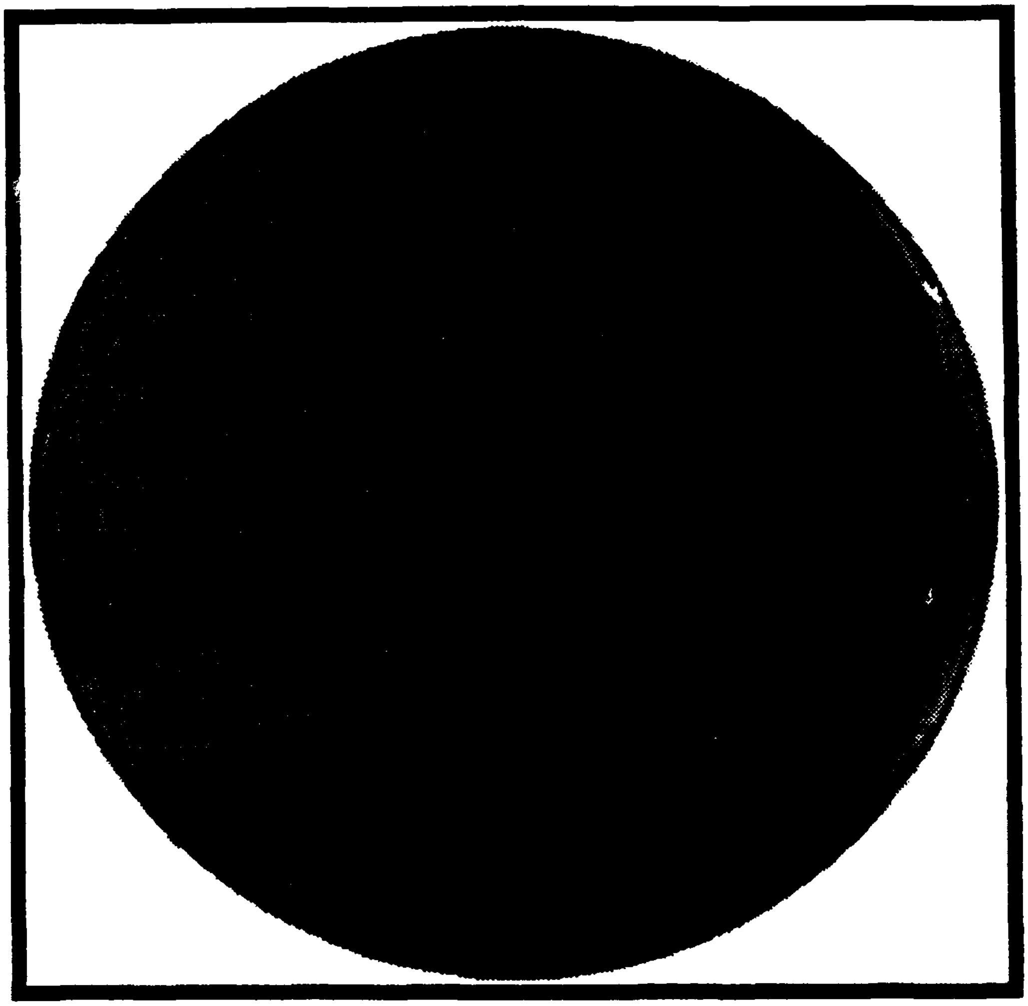

Figure 5-3. Satellite View

3. Mercator View

The Mercator View shows a map of the world as seen in Figure 5-4. The sun's relative plot is depicted as a yellow transparent ellipse on the map. This ellipse does not attempt to represent day/night transitions. Each satellite will have its sub-satellite point plotted in its color every time the satellite's position is updated. Satellite Parameters discusses how to turn plotting on.

C. PARAMETER WINDOW

The Parameter Window displays names and values of parameters for the Current Satellite and the SMET system. The user can change the value of many parameters with mouse and keyboard inputs. Distance and velocity parameters are displayed in one of three types of units. The three types are canonical (DU, DU/TU), metric (km, km/sec) and english (NM, ft/sec).

1. Satellite Parameters

There are five sets of Satellite Parameters - Classical, Secondary, Position, Options and Combined. The Classical set includes the parameters that describe the shape and orientation of the orbit. Most of these parameters can be changed by the user. The Secondary set also includes parameters that describe the shape of the orbit. Most of these parameters are actually computed by SMET for display and cannot be changed by the user. The Position set includes parameters that describe the satellites position in various coordinate systems. These parameters are also computed and cannot be changed by the user. The Options set is a set of parameters that SMET uses. The Combined set includes all the members of the other sets listed above. All parameter sets display the Current Satellite's name at the top. Figure 5-5 shows the Combined set of satellite parameters with the system parameters listed at the bottom.

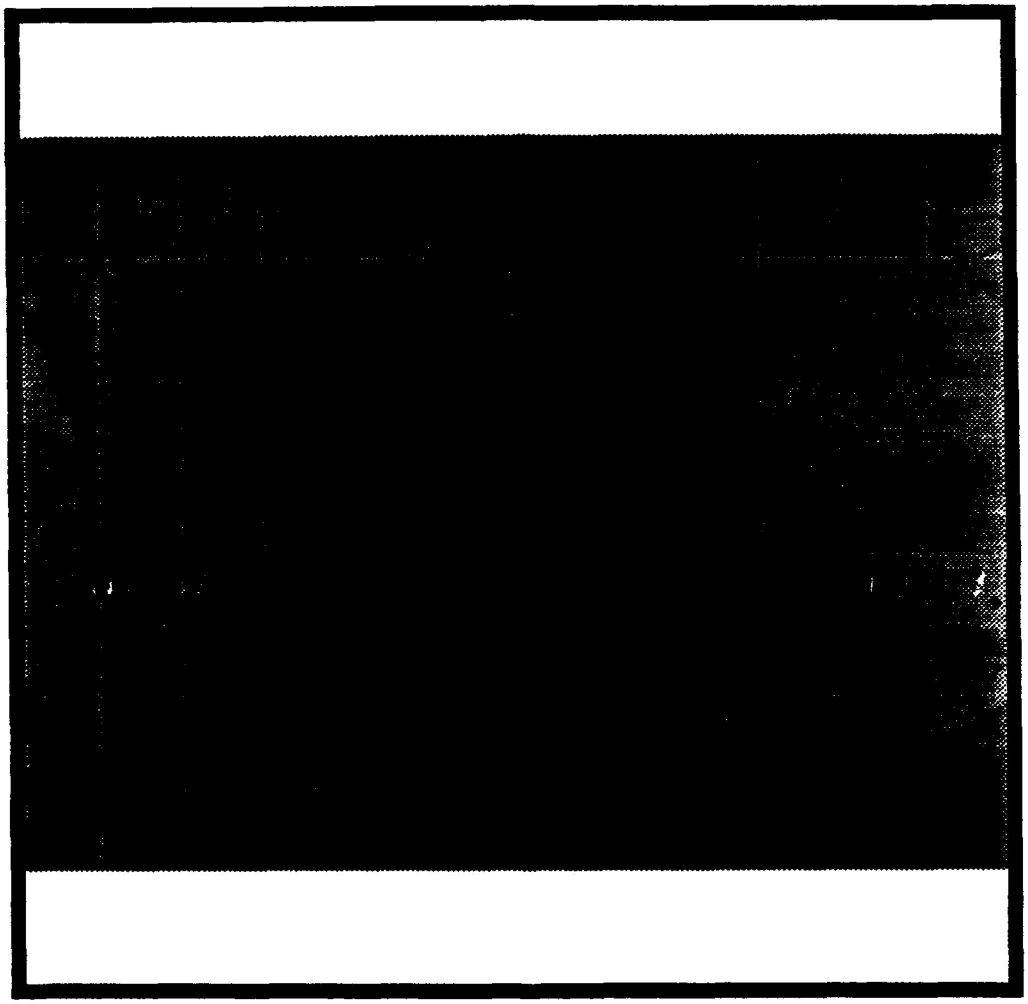

Figure 5-4. Mercator View

Satellite Parameters:

Name Molniya Orbit
Semi-Major 26561.78 km
Semi-Minor 17568.97 km
Eccentricity 0.75
Inclination 63.43
Ascending Node 59.55
Arg of Periapsis 270.00
True Anomaly 180.97
Epoch 25:54:00
Period 11:58:02
Apogee 46483.12 km
Perigee 6640.45 km
Range 46463.18 km
Velocity 1.47 km/sec
X -18308.79 km
Y 9851.25 km
Z 41551.97 km
P -46456.53 km
Q -786.01 km
Latitude 63.42 N
Longitude 150.09 W
Color Red
Fill Off
Vectors On
Plot On

System Parameters:

Clock On
Current Time 44:30:00
Clock Speed 00:30:00
Detailed Maps On
Political Borders On
Perturbations Off
Auto OmniViewer On
Viewer X 46483.12 km
Y 6378.15 km
Z 6378.15 km
Units Metric

Figure 5-5. Parameter Window

a. Classical Set

The Classical set of satellite parameters includes the semi-major axis, semi-minor axis, eccentricity, inclination, longitude of the ascending node, argument of periapsis, true anomaly, and time of epoch. The true anomaly is computed by SMET as a function of the elapsed time since epoch. It cannot be changed by the user. All other parameters can be adjusted by the user as described below.

b. Secondary Set

The Secondary set of satellite parameters includes the perigee distance, apogee distance, time of epoch, orbital period, current range and velocity. Except for the time of epoch, all of these parameters are computed by SMET and cannot be changed by the user.

c. Position Set

The Position set of satellite parameters displays the current position of the satellite in three different sets of coordinate systems. The X, Y and Z coordinates correspond with the geocentric-equatorial coordinate system displayed in the OmniViewer window. The P and Q coordinates correspond with the perifocal (P-Q-W) coordinate system. Both of these sets of coordinates are displayed in the current unit type (ex. metric). The Latitude and Longitude coordinates correspond with the earth-fixed coordinate system seen in the Mercator View window. None of these parameters can be change by the user.

d. Options Set

The Options set include the color of the satellite and its ellipse, if the orbital ellipse should be filled or empty, if the normal and perigee vectors should be displayed and if the satellite should be plotted on the Mercator map. All of these parameters can be changed by the user.

e. Combined Set

The Combined set includes all the members of the other sets listed above.

2. SMET System Parameters

SMET system parameters are listed below the set of satellite parameters. This set of parameters includes the clock, current time, clock speed, detailed maps, political borders, perturbations, Automatic OmniViewer, OmniViewer X, Y, and Z, and units. The clock parameter is used to start and stop the clock. Current time displays the elapsed time in hours:minutes:seconds format. It uses 00:00:00 GMT on the day of Vernal Equinox as the reference point. Clock speed allows the user to set the number of minutes between position updates. It is initially set to 30 minutes. The detailed maps parameter allows the user to switch between detailed and coarse globe and map data. The political borders parameter allows the user to turn off the display of political borders on the globe and map. Both of these options allow a user with a large number of satellites to speed up SMET by drawing less detailed images. They are both initially set to on. The perturbations parameter turns on the regression of the line of nodes and the rotation of the line of apsides. This is initially set to off. Automatic OmniViewer positioning can be turned on and off. It is initially set to on. If any of the OmniViewer coordinates (X, Y or Z) are changed, Automatic OmniViewer positioning is turned off. The OmniViewer coordinates are geocentric-equatorial coordinates. They are displayed in the current unit type. The units parameter displays the current unit type (canonical, metric or english). It is initially set to canonical. All system parameters can be changed by the user.

3. Parameter Input

A parameter in the Parameter Window is selected by placing the cursor over it and pressing the middle mouse button. If it can be changed, a dialog box or menu will appear. Dialog boxes are answered with keyboard entries. All keyboard entries must be entered in the current unit type (e.g., metric). A menu is answered by selecting an entry and pressing the right button. Parameters that toggle between "On" and "Off" will change with no other input required. Similarly, the units parameter will cycle through "Canonical", "Metric" and "English" each time it is selected. Certain parameters are computed by SMET. If one of these parameters is selected, the user is told that it cannot be changed.

SMET also allows the user to adjust certain parameters with five dials on the dial box. These include the clock speed and four of the classical parameters. Figure 5-6 shows how the dials are arranged. The dials are used for user convenience and simple demonstrations of the manipulation of parameters. If no dial box is available, these variables can still be input by using the Parameter Window. Whenever a dial is turned, the appropriate parameter will be changed. The clock speed dial is initially calibrated to operate between zero and six hours. If a clock speed greater than six hours is desired, the user can enter it by using the Parameter Window.

A change to a satellite's classical elements will radically alter its position. Therefore, SMET erases all displayed plots for the Current Satellite and begins plotting again. Similarly, if the current time is changed, all plots for all satellites are erased.

D. SATELLITE LIST WINDOW

The Satellite List Window, seen in Figure 5-7, lists all of the satellites in the system. The Current Satellite is listed at the top of the list. The user changes the Current Satellite by placing the cursor over the desired satellite and pressing the middle mouse button. The other satellites will shift down one position as necessary to reorganize the list.

E. MENU COMMANDS

When the user presses the right hand button on the mouse, a menu will appear near the position of the cursor. A menu item is selected by pressing the right hand mouse button again while the cursor is over a particular command or option. Certain menu items will have an arrow to the right. These items indicate that another menu (a rollover menu) will appear if the cursor is moved to the right of that menu item. Pressing the right hand mouse button while it is over the menu title or at any time that it is not over a menu item will abort the process of making a menu selection.

The main menu allows the user to add a satellite, maneuver the current satellite, delete the Current Satellite, save satellites' parameters into a file, take a picture, choose the set of satellite parameters to display in the Parameter Window, list the user instructions or exit the program.

1. Add Satellite

The Add Satellite command has three options that appear in a rollover menu. The user can define an orbit's parameters, define the parameters for a launch, or read a file of previously saved satellites.

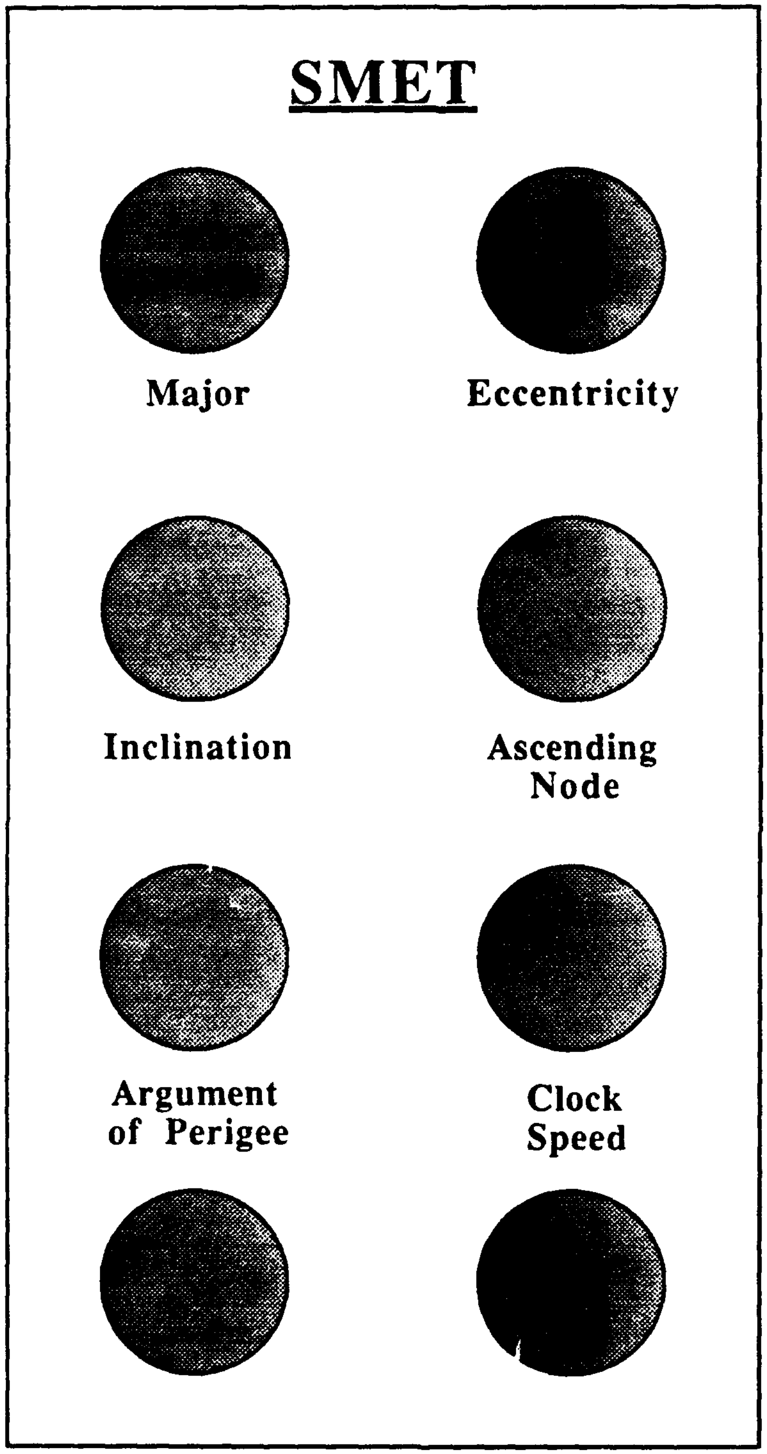

Figure 5-6. Dial Box

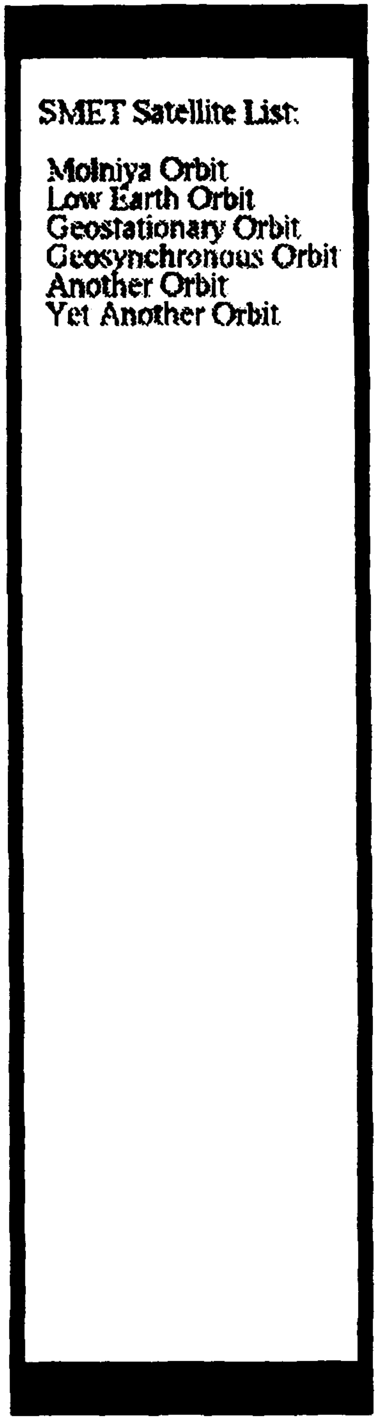

Figure 5-7. Satellite List Window

a. Define Orbit

Defining an orbit's parameters also has a rollover menu. Here the user is given the option of selecting from three pre-defined orbits (a Low Earth orbit (300 NM), a Molniya orbit, and a Geosynchronous orbit). The user also can select "Input Parameters". This allows the user to enter the classical parameters of the desired orbit through a series of dialog boxes. Parameters must be entered in the current unit type (e.g. km, if in metric). The user is asked to enter a name for the new satellite in all cases. It will be given a randomly selected color and have plotting, fill, and vectors turned off. The new satellite becomes the Current Satellite.

b. Define Launch

The Define Launch command also initiates a series of dialog boxes. The user is asked to enter a name for the new satellite and burnout parameters related to the launch. These parameters include the time at which burnout occurs (not to be confused with the length of the rocket burn), the altitude above the surface of the earth (in the current unit type, e.g. km), the velocity (in the current unit type, e.g. km/sec), the earth-fixed burnout latitude and longitude (±degrees), the flight path angle and the launch azimuth. If the time of burnout is earlier than the current time, the satellite will be placed at a position where it would have been had the launch been executed at the appropriate time. If not, a launch record is saved until the appropriate time of execution. When executing a launch, SMET will add in the appropriate earth rotation correction. The new satellite will be given a randomly selected color and have plotting, fill, and vectors turned off. It will become the Current Satellite. See Chapter III for more information.

c. Read File

The Read File command will read a satellite file previously saved using the Save Satellites command discussed below.

2. Maneuver Satellite

The Maneuver Satellite command initiates a sequence of dialog boxes. The user is asked to enter parameters for a maneuver of the Current Satellite. The first parameter is the true anomaly at which the maneuver should occur. If there are any maneuvers pending for that satellite, the user will be asked to enter the number of degrees of travel following a previous maneuver. When a previous maneuver is executed, this quantity will be added to the true anomaly at the end of the previous maneuver. For example, if 900 is entered the new maneuver will occur two and a half revolutions after the previous maneuver occurs. There is no limit to the number of maneuvers which can be added to a satellite's queue. Maneuvers will be executed in the order in which they are entered. A satellite does not have to be the Current Satellite at the time of the maneuver.

After the first entry the user is asked to choose between two methods of entering the parameters of a maneuver. The first option is to enter the magnitude of the velocity change along with an associated pitch and yaw. The user enters the magnitude in the current units (e.g., km/sec). The pitch and yaw are entered in degrees. Pitch is a right-hand rotation about the satellite's z-axis (thumb pointing towards the earth). Yaw is a right-hand rotation about the satellite's y-axis (thumb pointing opposite to the orbital plane's normal vector). SMET uses the magnitude, yaw and pitch of the velocity change to compute the changes to the magnitude of the velocity, flight path angle and heading angle of the satellite at the time of the maneuver. The second option for entering a maneuver

is to enter the changes to the magnitude of the velocity, flight path angle and heading angle directly. This option is useful to users who are more interested in the affects of a maneuver than in the total Δv required. See Chapter IV for more information.

3. Delete Satellite

The Delete Satellite command will remove the Current Satellite from the system permanently. The second satellite listed in the Satellite List Window becomes the Current Satellite.

4. Save Satellites

The Save Satellites command will first ask the user for a file name. The user may enter any Unix path and file name. If the file name is not unique, the old file will be overwritten. The user will then be asked for a number of satellites to be saved. Satellites are saved from in the order that they appear in the Satellite List Window. If the user only desires a specific subset of the satellites in the system, each satellite desired should first be selected in sequence until they are at the top of the list. Any file saved using this procedure can be read by Read File command discussed above. Saving satellites does not remove them from the system.

5. Take Picture

The Take Picture command has four options that will appear in a rollover menu. All options request a file name. The first two options will save an sgi-rgb formatted file. These files can be shown and manipulated with several image tools available on the IRIS. These image tools include showsgi, shrinksgi and others. They are available in the imagetool subdirectory of SMET's directory. The first option saves the entire screen as it appears. The second saves the entire screen after the background has been changed to white and white

items have been changed to gray. The third and fourth options will save the entire screen or the Main Window in Encapsulated PostScript (EPS) format. These files can be printed on a PostScript laserwriter or read by any software capable of reading EPS images. SMET will remain frozen for 30 to 90 seconds while the image is being saved.

6. Satellite Parameters

The Satellite Parameters command has a rollover menu. This menu allows the user to select which of the five sets of satellite parameters will be displayed in the Parameter Window.

7. Instructions

The Instructions command will temporarily halt SMET and display the user instructions.

8. Exit

The Exit command will quit SMET. The user is offered an opportunity to save any satellites desired. Any satellites in the system that have not been saved will be lost.

VI. PROGRAM MODIFICATIONS

A. GENERAL

There are several dummy function calls entered into SMET's source code. These dummy procedures can be used to make minor modifications to SMET without making major changes to the code. This capability is directed primarily at thesis students requiring a specific simulation. All of the functions described below are included in the file smet_user.c which is included in Appendix C. This file can be modified by novice C programmers for desired modifications to SMET. Other smet_xxx.c files should not be changed by inexperienced programmers. Modified versions of SMET should have the name altered to prevent confusion with the original SMET.

B. USER PROGRAM INITIALIZATION ROUTINE

The procedure **user_init** can be used to read in satellite files and initialize program variables to values different than the default value.

C. USER SATELLITE INITIALIZATION ROUTINE

The procedure **user_new_sat** can be used to modify any satellite variables to values different than the default value. The SMET satellite structure provides four unused variables (user_float1, user_float2, user_int1 and user_int2) which can be used in a modified program for such things as orbital perturbations or ASAT simulations.

D. USER PROGRAM UPDATE ROUTINES

SMET provides two procedures that can be used to check for an exit condition or a condition for executing a launch or maneuver. Procedures **user_pre_update_satellites** and **user_post_update_satellites** can also be used to check the elapsed time or to compare the positions of the satellites. One routine is called before all of the satellites are updated and the other is called after.

E. USER SATELLITE UPDATE ROUTINES

SMET provides two procedures that can be used to perturb an orbit or delete a satellite based on a given set of parameters. The procedure **user_pre_orbital_parameters** is called before the satellite's changing orbital parameters are updated. The procedure **user_post_orbital_parameters** is called after the changing parameters have been updated. Both procedures take a satellite and the time as arguments.

VII. SUMMARY, RECOMMENDATIONS AND FUTURE DIRECTIONS

SMET provides beginning students with the ability to learn the difficult concepts of orbital mechanics with greater ease. Satellite maneuvers, which are not covered in introductory courses in great detail, can now be taught with simple presentations. Additionally, a means of generating simulations is available for more advanced students.

In order to achieve the maximum benefit of this project, the Space Systems Academic Group should purchase an IRIS graphics workstation capable of running SMET. Several students in the Space Systems Operations Curriculum have already received instruction in programming the IRIS. Already, one student is creating a smet_user module for a SMET based ASAT simulation. As a start, modules could be written for standard transfer orbit maneuvers or perturbations not currently modeled. Space Systems Engineering students could also benefit from the use of a CAD system or other engineering software available for the IRIS. As an interim measure, the Space Systems Academic Group should negotiate limited access to an IRIS belonging to another department for the development of class presentations.

It is hoped that the software will be used elsewhere and continue to evolve. Future enhancements not currently provided include the ability to read NAVSPASUR Charlie Element formatted files for entering satellites, an illumination model to represent the sun, a more accurate representation of perturbations and a capability to automatically compute orbit transfer or intercept requirements.

APPENDIX A - COMMON VARIABLES

Variable Name	Formula Name	Program Name	
Semi-Major Axis	a	major	Description of conic section
Eccentricity	e	ecc	Description of conic section
Semi-Latus Rectum	p	p	Description of conic section
Range	r	range	Distance from the center of the earth to the satellite
Velocity	v	velocity	Magnitude of the velocity of the satellite
Inclination	i	incl	Angle between the plane of the equator and the plane of the orbit
Longitude of the Ascending Node	h	asc	Angle in the equatorial plane between the right ascension of the sun at Vernal Equinox and the right ascension of the satellite
Argument of Perigee	g	peri	Angle in the orbit plane from the line of nodes to perigee
True Anomaly	θ	anom	Angle in the orbit plane between perigee and the satellite
Eccentric Anomaly	E	E	Angle in the orbit plane between perigee and a "projection" of the satellite's position onto a circumscribed auxiliary circle
Mean Anomaly	M	M	The product of the mean motion and elapsed time
Epoch	t_o	epoch	Time of perigee passage
Heading Angle	ψ		Angle in the local horizon plane between the line through the satellite and the North point and the line of motion measured
Flight Path	γ	f_path	Angle in the orbit plane measured upward from the local horizon plane to the line of motion
Argument of Longitude	ω		Angle in the equatorial plane between the line of nodes and the meridian of the satellite
Argument of Latitude	u	u	Angle in the orbit plane between the line of nodes and the satellite
Latitude	ϕ	lat	Latitude of the subsatellite point
Longitude	λ	lon	Earth-fixed Longitude of the subsatellite point
Gravitational Parameter	μ		Constant for computation of earth orbits ($1\ DU_{\oplus}^3/TU_{\oplus}^2 = 1.407647 \times 10^{16}\ ft^3/sec^2$)

APPENDIX B - VIDEO RECORDING AND PRINTING

The following instructions apply to the implementation of SMET in the Graphics and Video Laboratory located in the Computer Science Department of the Naval Postgraduate School.

Video Recording - With Professor Zyda's permission, an S-VHS or a standard VHS tape can be recorded while running SMET on IRIS1. This tape can then be edited and dubbed as necessary for presentations. Note that some resolution will be lost because standard video signals have a lower resolution than the IRIS. The top and bottom edges of the image will also be clipped because of the different aspect ratios. The YEM scan converter should be turned on in order to record. The video recorder will record everything appearing on the video monitor.

Printing EPS Files - EPS files are saved using one of the EPS options of the Take Picture command. These files can be printed on any PostScript laser printer. The file can be printed with the following command: "lpr -Pps2 filename". The file can also be imported into other programs such as Framemaker for incorporation into other documents as was done in the preparation of this document.

APPENDIX C - SMET PROGRAM LISTING

SMET consists of the following files:

smet.h (p. 54 - 59) contains all constants and structures used by SMET. Also has all forward function, procedure and global variable declarations.

smet_vars.c (p. 60 - 64) defines all global variables.

smet_addelete.c (p. 65 - 68) includes procedures to add, delete and switch satellites. Additionally, procedures to add and delete plots to satellite structures are included.

smet_axes.c (p. 69) includes procedure to draw the axes in the OmniView window.

smet_dialog.c (p. 70 - 72) includes procedures for dialog and message boxes.

smet_files.c (p. 73 - 77) includes procedures that read and write satellite files, as well as reading SMET data files and writing image files.

smet_globe.c (p. 78 -79) includes procedures to draw the globe in the OmniView and Satellite View windows.

smet_init.c (p. 80 - 82) includes SMET initialization procedures.

smet_kepler.c (p. 83 - 85) includes procedures that update a satellite's orbital parameters.

smet_launch.c (p. 86 - 87) includes procedures to launch a satellite.

smet_main.c (p. 88 - 89) is the heart of SMET. It includes the main execution loop.

smet_maneuver.c (p. 90 - 95) includes procedures to maneuver a satellite.

smet_map.c (p. 96 -97) includes procedures to draw the map in the Mercator View window.

smet_orbit.c (p. 98) includes procedure to draw a satellite's orbital path in the OmniView window.

smet_param.c (p. 99 - 104) includes procedures that handle Parameter window events.

smet_queue.c (p. 105 - 109) includes procedures read the queue and handle events not covered in other files.

smet_sat.c (p. 110 - 113) includes procedures to draw a satellite in the OmniView window.

smet_user.c (p. 114) includes dummy procedures discussed in Chapter VI.

smet_windows.c (p. 115 - 123) includes procedures that draw the various windows.

makefile (p. 124) can be used to compile SMET in the NPS Computer Science Department Graphics and Video Laboratory. Files in the imagetools sub-directory will also be required.

smet_data.d (not included) is a binary data file necessary to draw the globe and map.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*      smet.h                                                     */
/*      by Carlos I. Noriega                                       */
/*                                                                 */
/*      This file contains the global variables, constants and     */
/*        structures used by the SMET demonstrator.                */
/*                                                                 */
/*      Written - September 3, 1990                                */
/*      Modified - September 3, 1990                               */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <math.h>                           /* math functions file    */
#include <string.h>                         /* string routines        */
#include "gl.h"                             /* graphics library  file */
#include "device.h"                         /* device header files    */
#include "stdio.h"                          /* I/O routines           */
#include "fmclient.h"                       /* IRIS font manager      */
#include "imagesupport/image_types.h"       /* for picture saving     */
#include "imagesupport/image_funcs.h"       /* for picture saving     */

#define FINEPOINTS  12843
#define COARSEPOINTS 3322
#define BORDERPOINTS 1407

#define OMNISATMERC 1   /*   Which view is in which window?          */
#define OMNIMERCSAT 2   /*   These constants used by switch_windows() */
#define SATMERCOMNI 3
#define SATOMNIMERC 4
#define MERCOMNISAT 5
#define MERCSATOMNI 6

#define EARTH        8  /*  Color Codes                   */
#define COAST        9
#define BORDER      10
#define LATLONG     11
#define SUN         12
#define SAT_RED     13
#define SAT_GREEN   14
#define SAT_YELLOW  15
#define SAT_BLUE    16
#define SAT_MAGENTA 17
#define SAT_CYAN    18
#define SAT_WHITE   19
#define SAT_GRAY    20
#define MAX_COLORS  21

#define CANONICAL   0   /*  Unit Codes                    */
#define METRIC      1
#define ENGLISH     2

#define DISTANCE  1     /*  for parameter type field      */
#define SPEED     2
#define DEGREE    3
#define LAT       4
#define LON       5
#define TIME      6
#define FLAG      7
#define FLOAT     8
#define STRING    9
#define COLORCODE 10
#define UNITCODE  11
```

```
#define ROTATION_RATE      4.37526951286e-3 /*  earth rate in rads/min  */
#define TU                 13.44686457      /*  minutes per TU          */
#define DU_METRIC          6378.1459997     /*  KM per DU               */
#define DU_ENGLISH         3443.9227864     /*  NM per DU               */
#define DU_PER_TU_METRIC   7.9053682830     /*  KM/SEC per DU/TU        */
#define DU_PER_TU_ENGLISH  25936.24764      /*  FT/SEC per DU/TU        */


#define PI                 3.14159265359    /*  Not mom's apple ...     */
#define TO_DEG             57.2957795132    /*Convert degrees to radians*/
#define TO_RAD             1.74532925199e-2 /*Convert degrees to radians*/
#define DEG30              0.523598775598   /*  30 degrees in radians   */
#define DEG60              1.04719755120    /*  60 degrees in radians   */
#define DEG90              1.57079632679    /*  90 degrees in radians   */
#define DEG120             2.09439510239    /*  120 degrees in radians  */
#define DEG180             3.14159265359    /*  180 degrees in radians  */
#define DEG270             4.71238898038    /*  270 degrees in radians  */
#define DEG360             6.28318530718    /*  360 degrees in radians  */


#define J2                 1.08264e-3       /*  perturbation constant   */
#define SUN_INCL           -23.454*TO_RAD   /*  pseudo-incl for plots   */
#define SUN_TU             83711.26224      /*  minutes per Solar TU    */


#define CLASSICAL   1     /*  parameter display options    */
#define SECONDARY   2
#define POSITIONS   3
#define OPTIONS     4
#define COMBINED    5

#define LETTER   18      /* font size  and line spacing  */
#define LINE     22

#define ERROR  -9999

#define max(x, y)        (x > y) ? x : y        /* return maximum value */
#define nonzerop(x)      fabs(x) > 0.00001      /* ~ float != 0.0       */
#define zerop(x)         fabs(x) < 0.00001      /* ~ float == 0.0       */
```

```
typedef struct {
        float x[FINEPOINTS];
        float y[FINEPOINTS];
        float z[FINEPOINTS];
        int pen[FINEPOINTS];
        } fine_data;


typedef struct {
        float x[COARSEPOINTS];
        float y[COARSEPOINTS];
        float z[COARSEPOINTS];
        int pen[COARSEPOINTS];
        } coarse_data;


typedef struct {
        float x[BORDERPOINTS];
        float y[BORDERPOINTS];
        float z[BORDERPOINTS];
        int pen[BORDERPOINTS];
        } border_data;


typedef struct  {       /*  Displayable parameters       */
        char *name;             /* for dialog boxes       */
        int type;               /* DISTANCE, SPEED, TIME */
        char *value;            /* cast this to whatever */
        } Par;


typedef struct  {       /*  Block of parameters        */
        int numparams;          /* number of parameters  */
        Par *params;            /* list of parameters    */
        } ParBlock;


typedef struct  {       /*  Window structure             */
        int orgx,orgy,          /* lower left corner     */
            sizex,sizey,        /* window size           */
            gid;                /* window grahics ID     */
        } window_struct;


typedef struct launch_struct { /*  Launch structure               */
        char    name[30];       /*  satellite name                 */
        float   time,           /*  when burnout will occur        */
                v,              /*  burnout velocity               */
                alt,            /*  burnout altitude               */
                lat,            /*  earth-fixed latitude           */
                lon,            /*  earth-fixed longitude          */
                elev,           /*  elev angle - above horizon     */
                azim;           /*  azimuth angle                  */
        struct launch_struct *next;/*  the next launch             */
        } launch_struct;
```

```
typedef struct plot_struct { /*  Plot structure */
        float lat,lon;            /* a plot           */
        struct plot_struct *next; /* the next plot    */
        } plot_struct;


typedef struct man_struct { /*  Maneuver structure                    */
        int     normal_maneuver;/*  flag for deltav,yaw,pitch   */
        float   time,           /*  when maneuver will occur    */
                anom,           /*  true anomaly of occurance   */
                deltav,         /*  Change in velocity          */
                yaw,            /*  yaw or heading change       */
                pitch;          /*  pitch or flight path change */
        struct man_struct *next;/*  the next maneuver           */
        } man_struct;


typedef struct satellite {      /*  Satellite structure                  */
        char    name[30];       /*  satellite name                       */
        float   major,          /*  semi-major axis of orbital ellipse   */
                minor,          /*  semi-minor axis of orbital ellipse   */
                ecc,            /*  eccentricity of orbital ellipse      */
                incl,           /*  inclination of orbital ellipse       */
                asc,            /*  longitude of the ascending node      */
                peri,           /*  argument of periapsis                */
                anom,           /*  true anomaly at epoch                */
                epoch,          /*  time of epoch +ve (if after 0000Z)   */
                E,              /*  eccentric anomaly at epoch           */
                p,              /*  X-coordinate in orbital plane        */
                q,              /*  Y-coordinate in orbital plane        */
                x,              /*  X-coordinate in IJK system           */
                y,              /*  Y-coordinate in IJK system           */
                z,              /*  Z-coordinate in IJK system           */
                range,          /*  Sat distance from center of earth    */
                apogee_range,   /*  Sat distance at apogee               */
                perigee_range,  /*  Sat distance at perigee              */
                lat,            /*  Satellite latitude                   */
                lon,            /*  Satellite longitude                  */
                period,         /*  Satellite period (one-revolution)    */
                velocity,       /*  current satellite velocity           */
                asc_change,     /*  pertrubation constant                */
                peri_change,    /*  pertrubation constant                */
                user_float1,    /*  for program modifications            */
                user_float2,    /*  for program modifications            */
                matrix[4][4];   /*  orbit's rotation matrix              */
        int     plot,           /*  are we plotting ?                    */
                fill,           /*  do we fill in the orbit ellipse?     */
                vectors,        /*  do we draw vectors on the ellipse?   */
                color,          /*  display color                        */
                user_int1,      /*  for program modifications            */
                user_int2;      /*  for program modifications            */
        plot_struct *plots;     /*  plots since plotting started         */
        man_struct *maneuver;   /*  next maneuver                        */
        struct satellite  *next;/*  pointer to next satellite            */
        } satellite;
```

```
/*  global  variable  declarations          */

extern
satellite header_sat,            /*  Header to sat list          */
          *current_sat;          /*  Current satellite           */

extern
launch_struct *pending_launches;/*  List of upcoming launches   */

extern
float clock,                     /*  Time elapsed                */
      clock_speed,               /*  time per minutes            */
      viewer_x,                  /*  Viewer X pos in Omniview    */
      viewer_y,                  /*  Viewer Y pos in Omniview    */
      viewer_z,                  /*  Viewer Z pos in Omniview    */
      viewer_range;              /*  Viewer Range in Omniview    */

extern
int  top_menu,                   /*  Menu ID                     */
     add_menu,                   /*  Menu ID                     */
     orbit_menu,                 /*  Menu ID                     */
     color_menu,                 /*  Menu ID                     */
     picture_menu,               /*  Menu ID                     */
     param_menu,                 /*  Menu ID                     */
     window_order,               /*  which-where ex. OMNISATMERC */
     omni_view_gid,              /*  gid of OMNI view window     */
     sat_view_gid,               /*  gid of SAT  view window     */
     merc_view_gid,              /*  gid of MERC view window     */
     unit_type,                  /*  CANONICAL, METRIC, AMERICAN */
     param_option,               /*  Which satellite parameters? */
     clock_running,              /*  is the clock running?       */
     fineflag,                   /*  Detailed Earth drawing?     */
     borderflag,                 /*  Draw political borders?     */
     auto_viewer,                /*  Auto pos for Omni view?     */
     perturbations;              /*  Do we do perturbations?     */

/*  Globe and map data records                                   */
extern fine_data fine2d_data, fine3d_data;
extern coarse_data coarse2d_data, coarse3d_data;
extern border_data border2d_data, border3d_data;

/* parameter block containing all displayable parameters         */
extern
ParBlock classicalparamblock,
         secondaryparamblock,
         positionparamblock,
         optionsparamblock,
         combinedparamblock,
         sysparamblock;
```

```
extern
char *instructions[28],          /* User instructions              */
     *color_names[MAX_COLORS],   /* Names of all the colors        */
     *unit_names[3],             /* Names of the Units             */
     *speed_unit[3],             /* Names of the speed units       */
     *distance_unit[3];          /* Names of the distance units    */

extern
float speed_factor[3],           /* Speed conversion factor        */
      distance_factor[3];        /* Distance conversion factor     */

extern                     /*   Set up all our window parameters   */
window_struct   backwin,
                mainwin,
                auxwin1,
                auxwin2,
                satlistwin,
                dialogwin,
                paramwin;

extern
float ident_matrix[4][4],        /* Nice constant to have laying around  */
      Color[MAX_COLORS][4];      /* Array of colors for c3f or c4f       */


/*  forward  function  declarations          */

draw_instructions();
add_launch();
ask_and_read_sat_file();
ask_and_write_sat_file();
add_maneuver();
add_satellite(int);
custom_satellite();
delete_satellite();
take_picture();
program_exit();
smet_param_option(int);
smet_sat_color(int);

float ask_time();
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                    */
/*      smet_vars.c                                                   */
/*      by Carlos I. Noriega                                          */
/*                                                                    */
/*      This file contains the global variables, constants and        */
/*        structures used by the SMET demonstrator.                   */
/*                                                                    */
/*      Written - September 3, 1990                                   */
/*      Modified - September 3, 1990                                  */
/*                                                                    */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

satellite header_sat,               /*  Header to sat list          */
          *current_sat = NULL;      /*  Pointer to list of sats     */

launch_struct *pending_launches = NULL;/*  Upcoming launches        */

float clock = 0.0,                  /*  Time elapsed in minutes     */
      clock_speed = 30.0,           /*  Time per cycle in minutes   */
      viewer_x,                     /*  Viewer X pos in Omniview    */
      viewer_y,                     /*  Viewer Y pos in Omniview    */
      viewer_z,                     /*  Viewer Z pos in Omniview    */
      viewer_range;                 /*  Viewer Range in Omniview    */

int   top_menu,                     /*  Menu ID                     */
      add_menu,                     /*  Menu ID                     */
      orbit_menu,                   /*  Menu ID                     */
      picture_menu,                 /*  Menu ID                     */
      color_menu,                   /*  Menu ID                     */
      param_menu,                   /*  Menu ID                     */
      window_order,                 /*  which-where ex. OMNISATMERC */
      omni_view_gid,                /*  gid of OMNI view window     */
      sat_view_gid,                 /*  gid of SAT  view window     */
      merc_view_gid,                /*  gid of MERC view window     */
      unit_type = CANONICAL,        /*  CANONICAL, METRIC, ENGLISH  */
      param_option = CLASSICAL,     /*  Which satellite parameters? */
      clock_running = TRUE,         /*  is the clock running?       */
      fineflag = TRUE,              /*  Detailed Earth drawing?     */
      borderflag = TRUE,            /*  Draw political borders?     */
      auto_viewer = TRUE,           /*  Auto pos for Omni view?     */
      perturbations = FALSE;        /*  Do we do perturbations?     */

/*  Globe and map data records                                      */

fine_data   fine2d_data, fine3d_data;
coarse_data coarse2d_data, coarse3d_data;
border_data border2d_data, border3d_data;
```

```
/* data structures for parameter display */

Par classical_list[] = {
        { "Name ", STRING, (char *)header_sat.name },
        { "Semi-Major ", DISTANCE, (char *)&(header_sat.major) },
        { "Semi-Minor ", DISTANCE, (char *)&(header_sat.minor) },
        { "Eccentricity ", FLOAT, (char *)&(header_sat.ecc) },
        { "Inclination ", DEGREE, (char *)&(header_sat.incl) },
        { "Ascending Node ", DEGREE, (char *)&(header_sat.asc) },
        { "Arg of Periapsis ", DEGREE, (char *)&(header_sat.peri) },
        { "True Anomaly ", DEGREE, (char *)&(header_sat.anom) },
        { "Epoch ", TIME, (char *)&(header_sat.epoch) } },

   secondary_list[] = {
        { "Name ", STRING, (char *)header_sat.name },
        { "Perigee ", DISTANCE, (char *)&(header_sat.perigee_range) },
        { "Apogee ", DISTANCE, (char *)&(header_sat.apogee_range) },
        { "Epoch ", TIME, (char *)&(header_sat.epoch) },
        { "Period ", TIME, (char *)&(header_sat.period) },
        { "Range ", DISTANCE, (char *)&(header_sat.range) },
        { "Velocity ", SPEED, (char *)&(header_sat.velocity) } },

   position_list[] = {
        { "Name ", STRING, (char *)header_sat.name },
        { "X ", DISTANCE, (char *)&(header_sat.x) },
        { "Y ", DISTANCE, (char *)&(header_sat.y) },
        { "Z ", DISTANCE, (char *)&(header_sat.z) },
        { "P ", DISTANCE, (char *)&(header_sat.p) },
        { "Q ", DISTANCE, (char *)&(header_sat.q) },
        { "Latitude ", LAT, (char *)&(header_sat.lat) },
        { "Longitude ", LON, (char *)&(header_sat.lon) } },

   option_list[] = {
        { "Name ", STRING, (char *)header_sat.name },
        { "Color ", COLORCODE, (char *)&(header_sat.color) },
        { "Fill ", FLAG, (char *)&(header_sat.fill) },
        { "Vectors ", FLAG, (char *)&(header_sat.vectors) },
        { "Plot ", FLAG, (char *)&(header_sat.plot) } },

   sysparamlist[] = {
        { "Clock ", FLAG, (char *)&clock_running },
        { "Current Time ", TIME, (char *)&clock },
        { "Clock Speed ", TIME, (char *)&clock_speed },
        { "Detailed Maps ", FLAG, (char *)&fineflag },
        { "Political Borders ", FLAG, (char *)&borderflag },
        { "Perturbations ", FLAG, (char *)&perturbations },
        { "Auto OmniViewer ", FLAG, (char *)&auto_viewer },
        { "Viewer X ", DISTANCE, (char *)&viewer_x },
        { "          Y ", DISTANCE, (char *)&viewer_y },
        { "          Z ", DISTANCE, (char *)&viewer_z },
        { "Units ", UNITCODE, (char *)&unit_type },
```

```
  combined_list[] = {
        { "Name ", STRING, (char *)header_sat.name },
        { "Semi-Major ", DISTANCE, (char *)&(header_sat.major) },
        { "Semi-Minor ", DISTANCE, (char *)&(header_sat.minor) },
        { "Eccentricity ", FLOAT, (char *)&(header_sat.ecc) },
        { "Inclination ", DEGREE, (char *)&(header_sat.incl) },
        { "Ascending Node ", DEGREE, (char *)&(header_sat.asc) },
        { "Arg of Periapsis ", DEGREE, (char *)&(header_sat.peri) },
        { "True Anomaly ", DEGREE, (char *)&(header_sat.anom) },
        { "Epoch ", TIME, (char *)&(header_sat.epoch) },
        { "Period ", TIME, (char *)&(header_sat.period) },
        { "Apogee ", DISTANCE, (char *)&(header_sat.apogee_range) },
        { "Perigee ", DISTANCE, (char *)&(header_sat.perigee_range) },
        { "Range ", DISTANCE, (char *)&(header_sat.range) },
        { "Velocity ", SPEED, (char *)&(header_sat.velocity) },
        { "X ", DISTANCE, (char *)&(header_sat.x) },
        { "Y ", DISTANCE, (char *)&(header_sat.y) },
        { "Z ", DISTANCE, (char *)&(header_sat.z) },
        { "P ", DISTANCE, (char *)&(header_sat.p) },
        { "Q ", DISTANCE, (char *)&(header_sat.q) },
        { "Latitude ", LAT, (char *)&(header_sat.lat) },
        { "Longitude ", LON, (char *)&(header_sat.lon) },
        { "Color ", COLORCODE, (char *)&(header_sat.color) },
        { "Fill ", FLAG, (char *)&(header_sat.fill) },
        { "Vectors ", FLAG, (char *)&(header_sat.vectors) },
        { "Plot ", FLAG, (char *)&(header_sat.plot) } } };

ParBlock  classicalparamblock = {
                9,                      /* numparams */
                classical_list          /* Par list  */
                 },
          secondaryparamblock = {
                7,                      /* numparams */
                secondary_list          /* Par list  */
                 },
          positionparamblock = {
                8,                      /* numparams */
                position_list           /* Par list  */
                 },
          optionsparamblock = {
                5,                      /* numparams */
                option_list             /* Par list  */
                 },
          combinedparamblock = {
                25,                     /* numparams */
                combined_list           /* Par list  */
                 },
          sysparamblock = {
                11,                     /* numparams */
                sysparamlist            /* Par list  */
                 };
```

```
char *instructions[28] =                    /*   User instructions          */
       {"                    SATELLITE MANEUVER EVALUATION TOOL","","","",
        "This program is designed to let the user view how the manipulation",
        "of different orbital parameters affect the path of a satellite.",
        "You will see the orbiting satellite from an omniscient point of",
        "point of view, the earth from the satellite's point of view and a",
        "mercator projection map with the track of the satellite marked.",
        "The user can also launch or maneuver satellites by use of a menu",
        "command.",
        "","" ,
        "INSTRUCTIONS:","",
        "Rightmouse:  Used to pop up menus to execute the desired command.",
        "",
        "",
        "Middlemouse: Used to select a parameter, a satellite or an ",
        "             Auxiliary Window.",
        "",
        "Dials:  The six upper dials will allow the user to adjust following",
        "        parameters.  The order is:   SEMI-MAJOR AXIS  ECCENTRICITY",
        "                                     INCLINATION      ASCENDING NODE",
        "                                     PERIGEE          CLOCK SPEED",
        "","",
        "Press the Rightmouse button to continue."},

     *color_names[MAX_COLORS] = { "Black", "Red", "Green", "Yellow", "Blue",
                                  "Magenta", "Cyan", "White","Earth", "Coast",
                                  "Border", "LatLong", "Sun","Red", "Green",
                                  "Yellow", "Blue", "Magenta", "Cyan", "White",
                                  "Gray"},

     *unit_names[3] = { "Canonical", "Metric", "English" },

     *speed_unit[3] = { "DU/TU", "km/sec", "ft/sec" },

     *distance_unit[3] = { "DU", "km", "NM"};

float speed_factor[3] = { 1.0, DU_PER_TU_METRIC, DU_PER_TU_ENGLISH },
      distance_factor[3] = { 1.0, DU_METRIC, DU_ENGLISH };
```

```
/*   Set up all our window parameters allowing for transfer     */
/*    to a different size screen system                         */

window_struct   /*  lower-left corner, window-size and dummy GID  */

        backwin  = { 0, 0, XMAXSCREEN, YMAXSCREEN, 0 },

        mainwin  = { XMAXSCREEN * 265/1279, YMAXSCREEN * 273/1023,
                     XMAXSCREEN * 750/1279, XMAXSCREEN * 750/1279, 0 },

        auxwin1  = { XMAXSCREEN * 315/1279, YMAXSCREEN * 23/1023,
                     XMAXSCREEN * 250/1279, XMAXSCREEN * 250/1279, 0 },

        auxwin2  = { XMAXSCREEN * 715/1279, YMAXSCREEN * 23/1023,
                     XMAXSCREEN * 250/1279, XMAXSCREEN * 250/1279, 0 },

        satlistwin  = { XMAXSCREEN * 1015/1279, YMAXSCREEN * 0/1023,
                        XMAXSCREEN * 264/1279, YMAXSCREEN * 1023/1023, 0 },

        dialogwin = { XMAXSCREEN * 270/1279, YMAXSCREEN * 620/1023,
                      XMAXSCREEN * 740/1279, YMAXSCREEN * 45/1023, 0 },

        paramwin = { XMAXSCREEN * 0/1279, YMAXSCREEN * 0/1023,
                     XMAXSCREEN * 265/1279, YMAXSCREEN * 1023/1023, 0 };


float ident_matrix[4][4] = { {1.0, 0.0, 0.0, 0.0} ,
                             {0.0, 1.0, 0.0, 0.0} ,
                             {0.0, 0.0, 1.0, 0.0} ,
                             {0.0, 0.0, 0.0, 1.0} },

      Color[MAX_COLORS][4] = {{0.000, 0.000, 0.000, 0.000},    /* BLACK        */
                              {1.000, 0.000, 0.000, 0.000},    /* RED          */
                              {0.000, 1.000, 0.000, 0.000},    /* GREEN        */
                              {1.000, 1.000, 0.000, 0.000},    /* YELLOW       */
                              {0.000, 0.000, 1.000, 0.000},    /* BLUE         */
                              {1.000, 0.000, 1.000, 0.000},    /* MAGENTA      */
                              {0.000, 1.000, 1.000, 0.000},    /* CYAN         */
                              {1.000, 1.000, 1.000, 0.000},    /* WHITE        */
                              {0.250, 0.800, 0.500, 0.000},    /* EARTH        */
                              {0.750, 0.200, 0.000, 0.000},    /* COAST        */
                              {0.500, 0.250, 0.000, 0.000},    /* BORDER       */
                              {0.750, 0.750, 0.750, 0.000},    /* LATLONG      */
                              {1.000, 0.800, 0.200, 0.400},    /* SUN          */
                              {0.750, 0.000, 0.000, 0.200},    /* SAT_RED      */
                              {0.000, 1.000, 0.000, 0.200},    /* SAT_GREEN    */
                              {1.000, 1.000, 0.000, 0.200},    /* SAT_YELLOW   */
                              {0.000, 0.500, 1.000, 0.200},    /* SAT_BLUE     */
                              {1.000, 0.000, 1.000, 0.200},    /* SAT_MAGENTA  */
                              {0.000, 1.000, 1.000, 0.200},    /* SAT_CYAN     */
                              {1.000, 1.000, 1.000, 0.200},    /* SAT_WHITE    */
                              {0.600, 0.600, 0.600, 0.200} };  /* SAT_GRAY     */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                    */
/*      smet_addelete.c                                               */
/*      by Carlos I. Noriega                                          */
/*                                                                    */
/*      This file contains the routines used by SMET to               */
/*      add and delete satellites, add and delete plots and           */
/*      switch and crash satellites.                                  */
/*                                                                    */
/*      Written - September 3, 1990                                   */
/*      Modified - September 3, 1990                                  */
/*                                                                    */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

/*  MAKE_NEW_SATELLITE                                             */
/*  Creates a new satellite and sets all default settings          */
/*     Asks user for name if none is provided.                     */

make_new_current_sat(name)
char *name;
{
  satellite *sat;

  sat = (satellite *)malloc(sizeof(satellite));
  sat->epoch = clock;
  if (strlen(name)) strcpy(sat->name,name);
  else dialog("Enter name of new satellite.", sat->name);
  sat->color = SAT_RED + rand() % 6;
  sat->fill = FALSE;
  sat->vectors  = FALSE;
  sat->plot = FALSE;
  sat->plots = NULL;
  sat->maneuver = NULL;
  sat->next = current_sat;
  header_sat.next = current_sat = sat;
  user_new_sat();
}
```

```
/*  ADD_SATELLITE                                              */
/*  Set the five basic parameters of the orbit and init-       */
/*     ialize the dials as appropriate.                        */

add_satellite(n)
int n;     /*  orbit type selected by pop up menu          */
{
  make_new_current_sat("");
  switch (n){
  case 1:                              /*  300 nm circular LEO   */
    current_sat->major = 1 + 300/distance_factor[ENGLISH];
    current_sat->ecc = 0.0;
    current_sat->incl = DEG30;
    current_sat->asc = DEG30;
    current_sat->peri = 0.0;
    break;
  case 2:                              /*  Molniya Orbit         */
    current_sat->major = pow(ROTATION_RATE*TU*2, -2.0/3.0);
    current_sat->ecc = 0.75;
    current_sat->incl = asin(sqrt(0.8));
    current_sat->asc = DEG60;
    current_sat->peri = DEG270;
    break;
  case 3:                              /*  Geosynchronous Orbit  */
    current_sat->major = pow(ROTATION_RATE*TU, -2.0/3.0);
    current_sat->ecc = 0.0;
    current_sat->incl = 0.0;
    current_sat->asc = 0.0;
    current_sat->peri = 0.0;
    break;
  case 4:                              /*  Custom Orbit          */
    ask_major();
    ask_ecc();
    ask_incl();
    ask_asc();
    ask_peri();
    break;
  }
  update_static_parameters(current_sat);
  update_sat_matrix(current_sat);
  reset_dials();               /*  set dials to current satellite  */
}
```

```
/*  DELETE_SATELLITE                                                   */
/*  Removes the current satellite from SMET                           */

delete_satellite()
{
  header_sat.next = current_sat->next;
  delete_plots(current_sat);
  free((char *) current_sat);
  current_sat = header_sat.next;
  if (current_sat != NULL) reset_dials();
}




/*  ADD_PLOT & DELETE_PLOTS                                            */
/*  Add a plot to the linked list or purge the list.                   */

add_plot(sat)
satellite *sat;
{
  plot_struct *new_plot;  /* link for new lat/lon  */

  new_plot = (plot_struct *)malloc(sizeof(plot_struct));
  new_plot->lat = sat->lat;
  new_plot->lon = sat->lon;
  new_plot->next = sat->plots;
  sat->plots = new_plot;
}

delete_plots(sat)
satellite *sat;
{
  plot_struct *temp;

  for (temp = sat->plots; sat->plots != NULL; temp = sat->plots)
  {
    sat->plots = sat->plots->next;
    free((char *) temp);
  }
}
```

```
/*  SWITCH_CURRENT_SAT                                          */
/*  Figures out which satellite was chosen from mouse y         */
/*  position.  Walk down list until you got it and switch'em.   */

switch_current_sat(y)
short y;
{
  int sat_number = (float)(satlistwin.sizey - y) / (float)LINE - 3,
      count;    /*  dummy loop counter  */
  satellite *walker;

  if (sat_number > 1)
  {
    walker = current_sat;
    for (count = 1; walker != NULL && count + 1 < sat_number; count++)
      walker = walker->next;
    set_next_to_current(walker);
  }
}


/*  CRASH_SATELLITE                                             */
/*  Set the desired sat to the current position so that         */
/*  you can do a delete satellite.  Called when range >= 1.     */

crash_satellite(sat, previous_sat)
satellite *sat, *previous_sat;
{
  char str[50];

  sprintf(str, "%s crashed into the earth and was deleted.",sat->name);
  message(str);
  set_next_to_current(previous_sat);
  delete_satellite();
  sat = previous_sat;
}


/*  SET_NEXT_TO_CURRENT                                         */
/*  Used by two previous procedures to switch the linked        */
/*  list of satellites.                                         */

set_next_to_current(previous)
satellite *previous;
{
  if (previous != NULL && previous->next != NULL && previous != &header_sat)
  {
    header_sat.next = previous->next;
    previous->next = previous->next->next;
    header_sat.next->next = current_sat;
    current_sat = header_sat.next;
    reset_dials();
  }
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*      smet_axes.c                                                  */
/*      by Carlos I. Noriega                                         */
/*                                                                   */
/*      This file contains the routines to draw the axes             */
/*         used by the SMET demonstrator.                            */
/*                                                                   */
/*      Written - September 3, 1990                                  */
/*      Modified - September 3, 1990                                 */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"


/*  DRAW-AXES                                           */
/*  draw the IJK axes at the center of the              */
/*     world coordinate system.  Remember that          */
/*     IJK is not aligned the same as the systems       */
/*     XYZ coordinate system.                           */

draw_axes()
{
  /*  go to white   */
  c3f(Color[WHITE]);

  /*  set the linewidth - what a comment !!  */
  linewidth(3);

  /*  label the I axis   */
  cmov(0.0, 0.2, 1.2);
  fmprstr("I");

  /*  label the J axis   */
  cmov(1.2, 0.2, 0.0);
  fmprstr("J");

  /*  label the K axis   */
  cmov(0.2, 1.2, 0.0);
  fmprstr("K");

  /*  draw the J axis    */
  move (1.0, 0.0, 0.0);
  draw(12.5, 0.0, 0.0);

  /*  draw the K axis    */
  move (0.0, 1.0, 0.0);
  draw(0.0, 12.5, 0.0);

  /*  draw the I axis    */
  move (0.0, 0.0, 1.0);
  draw(0.0, 0.0, 12.5);
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*      smet_dialog.c                                                */
/*      by Carlos I. Noriega                                         */
/*                                                                   */
/*      This file contains the routines to give the user messages    */
/*         and ask him questions.  It is used by the SMET            */
/*         demonstrator.                                             */
/*                                                                   */
/*      Written - September 3, 1990                                  */
/*      Modified - September 3, 1990                                 */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

/*  DIALOG & MESSAGE                                              */
/*  Use the dialogbox routines to send a message or ask a         */
/*  question.                                                     */

dialog(prompt,response)
char *prompt, *response;
{
  open_dialogbox();
  get_user_response(prompt,response);
  close_dialogbox();
}


message(prompt)
char *prompt;
{
  open_dialogbox();
  clearprompt(prompt);
  ringbell();
  sleep(3);
  close_dialogbox();
}
```

```
/*  DIALOGBOX Routines                                              */
/*  Uses the hidden dialogwin for all user messages.                */

open_dialogbox()          /* "POP" the box.  */
{
  winset(dialogwin.gid);
  ortho2(0.0,dialogwin.sizex, 0.0, dialogwin.sizey);
  loadmatrix(ident_matrix);
  singlebuffer();
  gconfig();
  winpop();
  zclear();
  qdevice(KEYBD);
}


clearprompt(prompt)       /* Clear the box and type the message.  */
char *prompt;
{
  c3f(Color[CYAN]);
  clear();
  c3f(Color[BLACK]);
  linewidth(2);
  recti(7, 7, dialogwin.sizex-7, dialogwin.sizey-7);
  cmov2i(12, 20);
  charstr(prompt);  charstr(" ");
}


close_dialogbox()         /* "PUSH" the box.  */
{
  zclear();
  winpush();
  unqdevice(KEYBD);
}
```

```
/*  GET_USER_RESPONSE                                          */
/*  A simple parser for user interaction.  Does not currently  */
/*  check if response string is shorter than fixed length of 40.*/
/*  Ignores events other than the keyboard.                    */

get_user_response(prompt,response)
char *prompt, *response;
{
    int cur_str_len,
        donetyping = FALSE;
    short c;
    char *str;
    Device dev;

    response[0] = '\0';
    cur_str_len = strlen(response);

    clearprompt(prompt);    /* display prompt */

    while(!donetyping) /* read till eof ('\n' or '\r') */
    if(dev = qread(&c))
    {
        if(dev != KEYBD) continue;        /* don't care */
        switch(c)
        {
            case '\027':        /* ^W sets cursor back to start */
                response[0] = '\0';
                cur_str_len = strlen(response);
                clearprompt(prompt);
                break;
            case '\n':
            case '\r':          /* carriage return or enter     */
                donetyping = TRUE;
                break;
            case 127:
            case '\b':          /*  backspace or delete         */
                if(cur_str_len) {
                    response[--cur_str_len] = '\0';
                    clearprompt(prompt);
                    charstr(response);  }
                else ringbell();
                break;
            default:            /*  add to string               */
                if(cur_str_len < 41) {
                    str = &response[cur_str_len];
                    response[cur_str_len++] = c;
                    response[cur_str_len] = '\0';
                    charstr(str);  }
                else ringbell();
                break;
        }
    }
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*      smet_files.c                                                 */
/*      by Carlos I. Noriega                                         */
/*                                                                   */
/*      This file contains the routines used by SMET to              */
/*         read and write files.                                     */
/*                                                                   */
/*      Written - September 3, 1990                                  */
/*      Modified - September 3, 1990                                 */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"


/*  READ_WORLD_FILE                                               */
/*  Reads earth data for map and globe from file in the           */
/*  current directory.                                            */

read_world_file()
{
  FILE  *fd;

  fd = fopen("smet_data.d","r");
  if(fd == NULL)
  {
    message("Unable to open file smet_data.d, quiting SMET");
    program_exit();
  }
  else
  {
    lseek(fd,0L,0);
    fread(&fine3d_data, sizeof(fine3d_data), 1, fd);
    fread(&fine2d_data, sizeof(fine2d_data), 1, fd);
    fread(&coarse3d_data, sizeof(coarse3d_data), 1, fd);
    fread(&coarse2d_data, sizeof(coarse2d_data), 1, fd);
    fread(&border3d_data, sizeof(border3d_data), 1, fd);
    fread(&border2d_data, sizeof(border2d_data), 1, fd);
    fclose(fd);
  }
}
```

```
/*  ASK_AND_READ_SAT_FILE & READ_SAT_FILE                               */
/*  Reads satellite data previously saved.                              */

ask_and_read_sat_file()
{
  char name[40];

  dialog("Enter the name of the satellite file.",name);
  read_sat_file(name);
}



read_sat_file(file_name)
char file_name[];
{
  FILE  *fd;
  satellite *sat;
  int   magic;
  char str[50];

  fd = fopen(file_name,"r");
  if(fd == NULL)
  {
    sprintf(str,"Unable to open file - %s.",file_name);
    message(str);
  }
  else
  {
    lseek(fd,0L,0);
    fread(&magic, sizeof(int), 1, fd);
    if (magic != 4567)                  /* ensure properly formatted file */
    {
    sprintf(str,"%s is not a satellite file.",file_name);
    message(str);
    }
    else
    {   /* read satellites until you get a read error   */
      sat = (satellite *)malloc(sizeof(satellite));
      while (fread(sat, sizeof(satellite), 1, fd))
      {
        sat->plots = NULL;
        sat->maneuver = NULL;
        sat->next = current_sat;
        current_sat = sat;
        sat = (satellite *)malloc(sizeof(satellite));
      }
      free((char *) sat);
      header_sat.next = current_sat;  /* make last one current   */
      reset_dials();
    }
    fclose(fd);
  }
}
```

```
/*  ASK_AND_WRITE_SAT_FILE & WRITE_SAT_FILE                    */
/*  Write satellite data into a file.                          */

ask_and_write_sat_file()
{
  char name[40], number[5];

  dialog("Enter the name of the satellite file.",name);
  dialog("Enter the number of the satellites saved.",number);
  write_sat_file(name,atoi(number));
}



write_sat_file(file_name,numsats)
char file_name[];
int numsats;
{
  FILE  *fd;
  satellite *sat;
  int c, magic = 4567;
  char str[50];

  fd = fopen(file_name,"w");
  if(fd == NULL)
  {
    sprintf(str,"Unable to open file - %s.",file_name);
    message(str);
  }
  else
  {
    lseek(fd,0L,0);
    fwrite(&magic, sizeof(int), 1, fd); /* ensure properly formatted file */
    sat = current_sat;
    for (c = 0; (c < numsats && sat != NULL); c++)
    {
      fwrite(sat, sizeof(satellite), 1, fd);
      sat = sat->next;
    }
    fclose(fd);
  }
}
```

```
/*  TAKE-PICTURE                                          */
/*  Use image library routines to write an image          */
/*  in the selected format.  Black and White are          */
/*  inverted for use in EPS format.                       */

take_picture(num)
int num;
{
  NPSIMAGE *img1,*img2, *img3;    /* temp ptr to images */
  char  filename[50],
        str[75];

  switch (num)
  {
  case 1:
        dialog("Enter the name for the RGB file -", filename);
        message("Saving the screen will take about 90 seconds.");
        refresh_main();
        save_window_as_sgi_rgbimage(filename,backwin.gid);
        break;
  case 2:
        dialog("Enter the name for the RGB file -", filename);
        message("Saving the screen will take about 90 seconds.");
        invert_bw();
        save_window_as_sgi_rgbimage(filename,backwin.gid);
        revert_bw();
        break;
  case 3:
        dialog("Enter the name for the EPS file -", filename);
        message("Saving the screen will take about 60 seconds.");
        invert_bw();
        img1 = read_window_fb_as_rgb_npsimage(backwin.gid,filename);
        img2 = shrink_npsimage(img1, 2);
        delete_an_image(img1);
        img3 = rotate_npsimage(img2);
        delete_an_image(img2);
        write_eps_image(filename, img3, 125);
        delete_an_image(img3);
        revert_bw();
        break;
  case 4:
        dialog("Enter the name for the EPS file -", filename);
        message("Saving the main window will take about 30 seconds.");
        invert_bw();
        img1 = read_window_fb_as_rgb_npsimage(mainwin.gid,filename);
        img2 = shrink_npsimage(img1, 2);
        delete_an_image(img1);
        write_eps_image(filename, img2, 125);
        delete_an_image(img2);
        revert_bw();
        break;
  }
}
```

```
/*  Support routines for TAKE_PICTURE                             */
/*                                                                */

refresh_main()
{
  switch(window_order)  /* refresh main window after message */
  {
    case OMNISATMERC:
    case OMNIMERCSAT: draw_omni_view(); break;
    case SATOMNIMERC:
    case SATMERCOMNI: draw_sat_view(); break;
    case MERCOMNISAT:
    case MERCSATOMNI: draw_merc_view(); break;
  }
}


invert_bw()
{
  Color[BLACK][0] = Color[BLACK][1] = Color[BLACK][2] = Color[BLACK][3] = 1.0;
  Color[WHITE][0] = Color[WHITE][1] = Color[WHITE][2] = Color[WHITE][3] =
  Color[SAT_WHITE][0] = Color[SAT_WHITE][1] = Color[SAT_WHITE][2] =
                                                  Color[SAT_WHITE][3] = 0.25;
  winset(backwin.gid);
  c3f(Color[BLACK]);
  clear();
  swapbuffers();
  draw_windows();
  draw_windows();  /* draw twice to ensure that proper buffers are showing */
}


revert_bw()
{
  Color[BLACK][0] = Color[BLACK][1] = Color[BLACK][2] = Color[BLACK][3] = 0.0;
  Color[WHITE][0] = Color[WHITE][1] = Color[WHITE][2] = Color[WHITE][3] =
  Color[SAT_WHITE][0] = Color[SAT_WHITE][1] = Color[SAT_WHITE][2] =
                                                  Color[SAT_WHITE][3] = 1.0;
  winset(backwin.gid);
  c3f(Color[BLACK]);
  clear();
  swapbuffers();
  draw_windows();
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                    */
/*      smet_globe.c                                                  */
/*      by Carlos I. Noriega                                          */
/*                                                                    */
/*      This file contains the routines used by SMET to               */
/*         draw the globe in a viewing window.                        */
/*                                                                    */
/*      Written - September 3, 1990                                   */
/*      Modified - September 3, 1990                                  */
/*                                                                    */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

/*  DRAW-GLOBE                                              */
/*  Assumes that a central disk normal to viewer drawn      */
/*  by calling routine.                                     */

draw_globe(rotation)
float  rotation; /*  amount of y-rotation of the globe     */
{
  pushmatrix();

  rot(rotation*TO_DEG,'y');  /*  turn the globe   */

  draw_coasts3d();
  draw_borders3d();
  draw_latlons3d();

  popmatrix();
}


/*  DRAW-COASTS3D                                           */
/*  Draw the desired data FINE or COURSE.                   */

draw_coasts3d()
{
  int c;

  linewidth(2);
  c3f(Color[COAST]);
  if (fineflag)
    for (c = 0; c < FINEPOINTS; c++)
      if (fine3d_data.pen[c])
        draw(fine3d_data.x[c], fine3d_data.y[c], fine3d_data.z[c]);
      else
        move(fine3d_data.x[c], fine3d_data.y[c], fine3d_data.z[c]);
  else
    for (c = 0; c < COARSEPOINTS; c++)
      if (coarse3d_data.pen[c])
        draw(coarse3d_data.x[c], coarse3d_data.y[c], coarse3d_data.z[c]);
      else
        move(coarse3d_data.x[c], coarse3d_data.y[c], coarse3d_data.z[c]);
}
```

```
/*  DRAW-BORDERS3D                                          */
/*  Draw the border data if desired.                        */

draw_borders3d()
{
  int c;

  linewidth(1);
  c3f(Color[BORDER]);
  if (borderflag)
    for (c = 0; c < BORDERPOINTS; c++)
      if (border3d_data.pen[c])
        draw(border3d_data.x[c], border3d_data.y[c], border3d_data.z[c]);
      else
        move(border3d_data.x[c], border3d_data.y[c], border3d_data.z[c]);
}



/*  DRAW-LATLONS3D                                          */
/*  Draw the latitude and longitude lines on the globe.     */

draw_latlons3d()
{
  int c;

  linewidth(1);
  c3f(Color[LATLONG]);
  pushmatrix();

  /*   draw the longitude lines    */
  pushmatrix();
  for (c = 1; c <= 12; c++)  /*  draw an arc every 30 degrees    */
  {
    rotate(300,'y');
    arc(0.0,0.0,1.0,-900,900);
  }
  popmatrix();

  /*   draw the latitude lines     */
  rotate(900,'x');      /*  flip over on its side     */
  circ(0.0,0.0,1.0);          /*  equator               */
  translate(0.0,0.0,-0.5);
  circ(0.0,0.0,0.866);        /*  30 degrees north      */
  translate(0.0,0.0,-0.366);
  circ(0.0,0.0,0.5);          /*  60 degrees north      */
  translate(0.0,0.0,1.366);
  circ(0.0,0.0,0.866);        /*  30 degrees south      */
  translate(0.0,0.0,0.366);
  circ(0.0,0.0,0.5);          /*  60 degrees south      */

  popmatrix();
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                    */
/*      smet_axes.c                                                   */
/*      by Carlos I. Noriega                                          */
/*                                                                    */
/*      This file contains the initialization routines                */
/*         used by the SMET demonstrator.                             */
/*                                                                    */
/*      Written - September 3, 1990                                   */
/*      Modified - September 3, 1990                                  */
/*                                                                    */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"


/*  INITIALIZE_SMET                                         */
/*  Open windows, set up menus, initialize queue,           */
/*      draw the user instructions.                         */

initialize_smet(argc,argv)
int argc;
char **argv;
{
  int c;
  fmfonthandle param_font;

  /*  initialize windows                                   */

  init_window(&backwin);        /* a black background                */
  init_window(&dialogwin);      /* the dialog window                 */
  init_window(&satlistwin);     /* where the satellite names are listed */
  init_window(&paramwin);       /* where the parameters are listed   */
  init_window(&mainwin);        /* the centered large window         */
  init_window(&auxwin1);        /* the lower left small window       */
  init_window(&auxwin2);        /* the lower right small window      */

  message("                                Loading SMET");

  /*  make the pop up menus                                */

  color_menu = defpup("Color %t %F | Red %x0 | Green %x1 | Yellow %x2 \
| Blue %x3 | Magenta %x4 | Cyan %x5 | White %x6 | Gray %x7", smet_sat_color);

  orbit_menu = defpup("Define Orbit %t %F | Low Earth | Molniya \
| Geosynchronous | Input Parameters", add_satellite);

  add_menu = defpup("New Satellite %t | Define Orbit %m | Define Launch %f \
| Read File %f", orbit_menu, add_launch, ask_and_read_sat_file);

  param_menu = defpup("Parameter Options %t %F | Classical | Secondary \
| Position | Options | Combined", smet_param_option);

  picture_menu = defpup("Picture Options %t %F | Color RGB Image (Regular) %x1 \
| Color RGB Image (Inverted) %x2  | Post Script - Entire Screen %x3  | Post Script \
- Main Window %x4", take_picture);

  top_menu = defpup("SMET menu %t | Add Satellite %m | Maneuver Satellite %f \
| Delete Satellite %f | Save Satellites %f | Take Picture %m | Satellite \
Parameters %m | Instructions %f | Exit %f ", add_menu, add_maneuver,
delete_satellite, ask_and_write_sat_file, picture_menu, param_menu,
draw_instructions, program_exit);
```

```
  /*  queue the necessary devices                              */

  qdevice(MIDDLEMOUSE);
  qdevice(RIGHTMOUSE);
  qdevice(DIAL2);
  qdevice(DIAL3);
  qdevice(DIAL4);
  qdevice(DIAL5);
  qdevice(DIAL6);
  qdevice(DIAL7);
  tie(MIDDLEMOUSE,MOUSEX,MOUSEY);
  qreset();  /*  initialize the queue   */

  /* dial limits: one minute to 6 hours  */
  setvaluator(DIAL3, (short)(clock_speed), 1, 360);



  /*  miscellaneous initializations                            */



  /* read data files for drawing globe and map                 */
  read_world_file();

  /* Initialize the satellite list                             */
  header_sat.next = NULL;

  /* read any sat files listed by user on startup              */
  for (c = 1; c < argc; c++) read_sat_file(argv[c]);

  /* Initialize font for windows                               */
  fminit();
  fmsetfont(fmscalefont(fmfindfont("Times-Roman"), LETTER));

  /* User Initialition Routine                                 */
  user_init();

  /*  put the different views into the default windows  */
  omni_view_gid = mainwin.gid;
  sat_view_gid = auxwin1.gid;
  merc_view_gid = auxwin2.gid;
  window_order = OMNISATMERC;


  /*  draw instructions for the user                           */
  draw_instructions();
}
```

```
/*  INIT-WINDOW                                        */
/*  Set up a window using the data from the given      */
/*       window structure                              */

init_window(win)
window_struct *win;   /*  window structure             */
{
  /*  set the pref position according to structure     */
  prefposition(win->orgx, win->orgx + win->sizex,
               win->orgy, win->orgy + win->sizey);

  win->gid = winopen("");  /*  create a window      */
  noborder();              /*  no edges or window manager pop ups  */
  RGBmode();               /*  RGB color specifications            */
  doublebuffer();          /*  no flickering                       */
  zbuffer(TRUE);           /*  hidden surface removal              */
  lsetdepth(0, 0x7fffff);  /*  get the largest Zbuffer resolution  */
  gconfig();               /*  attach settings to the current window */
  mmode(MVIEWING);         /*  put winod in viewing mode           */

  /*  go black and clear both buffers    */
  c3f(Color[BLACK]);
  clear();
  swapbuffers();
  clear();
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*      smet_kepler.c                                                */
/*      by Carlos I. Noriega                                         */
/*                                                                   */
/*      This file contains the routines to update orbital            */
/*         parameters.  It is used by the SMET demonstrator.         */
/*                                                                   */
/*      Written - September 3, 1990                                  */
/*      Modified - September 3, 1990                                 */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"


/*  UPDATE-STATIC-PARAMETERS                                      */
/*  See Chapter II of thesis.                                     */

update_static_parameters(sat)
satellite *sat;
{
  /*  update the semi-minor axis, period, perigee, apogee when    */
  /*  ecc or major  have changed due to a maneuver or user input  */
  sat->minor = sat->major * sqrt(1 - sat->ecc * sat->ecc);
  sat->period = pow(sat->major,1.5) * DEG360 * TU;
  sat->apogee_range = sat->major * (1 + sat->ecc);
  sat->perigee_range = sat->major * (1 - sat->ecc);
  update_perturb_constants(sat);
}



/*  UPDATE-PERTURB_CONSTANTS                                      */
/*  See Chapter II of thesis.                                     */

update_perturb_constants(sat)
satellite *sat;
{
  float perturb_constant = J2 / (pow(sat->major, 3.5) * pow((1 - sat->ecc * sat->ecc), 2))

  /* constant for regression of line of nodes  */
  sat->asc_change = - 1.5 * cos(sat->incl) * perturb_constant /TU;

  /* constant for rotation of line of apsides  */
  sat->peri_change = (3 - 3.75 * sin(sat->incl)*sin(sat->incl)) * perturb_constant /TU;
}
```

```
/*  UPDATE-CHANGING-PARAMETERS                                              */
/*   See Chapter II of thesis.                                              */

update_changing_parameters(sat, time)
satellite  *sat;
float time;
{
  float E, M;   /*  Eccentric and Mean anomalies        */

  if (clock_running && perturbations)
  {
    /* regression of line of nodes  */
    sat->asc = sat->asc + sat->asc_change * clock_speed;

    /* rotation of line of apsides  */
    sat->peri = sat->peri + sat->peri_change * clock_speed;
    update_sat_matrix(sat);
  }

  /* use M as initial approximation of eccentric anomaly */
  sat->E = M = fmod(((time - sat->epoch) * DEG360)/sat->period, DEG360);
  if (M < 0.0) sat->E = M = DEG360 + M;   /* ie. time < epoch  */

  /*  compute the Eccentric anomaly using Newtonian Approximation  */
  /* normally converges after 3 to 5 iterations at this accuracy   */
  do
  {
    E = sat->E;
    sat->E = E + (M - E + sat->ecc * sin(E))/(1 - sat->ecc * cos(E));
  } while (nonzerop(sat->E - E)); /*  repeat till almost zero  */

  /*  compute the true anomaly from the eccentric anomaly    */
  sat->anom = acos((cos(sat->E) - sat->ecc)/(1 - sat->ecc * cos(sat->E)));
  if (sat->E > DEG180) sat->anom = DEG360 - sat->anom;   /* get correct quadrant */

  /*  compute the distance to the satellite                 */
  sat->range = sat->major * (1 - sat->ecc * cos(sat->E));

  /*  compute the velocity of the satellite                 */
  sat->velocity = sqrt((2/sat->range) - (1/sat->major));

  /*  compute the planar coordinates of the satellite       */
  sat->p = sat->range * cos(sat->anom);
  sat->q = sat->range * sin(sat->anom);

  /*  compute the XYZ coordinates of the satellite.  Don't  */
  /*     forget that this is for an IJK system which is     */
  /*     different than the display systems XYZ system      */
  sat->y = sat->q*sat->matrix[0][0]+sat->p*sat->matrix[2][0]+sat->matrix[3][0];
  sat->z = sat->q*sat->matrix[0][1]+sat->p*sat->matrix[2][1]+sat->matrix[3][1];
  sat->x = sat->q*sat->matrix[0][2]+sat->p*sat->matrix[2][2]+sat->matrix[3][2];

  /*  compute the sat's lat and long for plotting adjustinf for earth spin  */
  sat->lat = asin(sat->z / sat->range);
  sat->lon = atan2(sat->y, sat->x) - fmod(time * ROTATION_RATE,DEG360);
  if (sat->lon < -DEG180) sat->lon = sat->lon + DEG360;

   /* save the plot if it is being plotted  */
  if (sat->plot) add_plot(sat);
}
```

```
/*  UPDATE-SAT_MATRIX                                        */
/*  compute the orbital plane transformation matrix          */
/*  lets let the graphics hardware do some of our            */
/*  tough calculations.   what a neat trick!                 */

update_sat_matrix(sat)
satellite *sat;
{
  loadmatrix(ident_matrix);
  rot(sat->asc*TO_DEG,'y');     /*  longitude of the ascending node     */
  rot(sat->incl*TO_DEG,'z');    /*  inclination of the orbital plane    */
  rot(sat->peri*TO_DEG,'y');    /*  argument of periapsis               */
  getmatrix(sat->matrix);
}


/*  ALT_UPDATE-SAT_MATRIX                                    */
/*  Use for multiprocessing when encorporated.               */
/*  Gets same matrix as update_sat_matrix.                   */
/*  Slightly different than matrix in Chapter 2 of           */
/*  thesis due to difference in coordinate systems.          */

alt_update_sat_matrix(sat)
satellite *sat;
{
  float cos_asc = cos(sat->asc),
        sin_asc = sin(sat->asc),
        cos_incl = cos(sat->incl),
        sin_incl = sin(sat->incl),
        cos_peri = cos(sat->peri),
        sin_peri = sin(sat->peri);

  sat->matrix[0][0] = -sin_asc*sin_peri + cos_asc*cos_peri*cos_incl;
  sat->matrix[0][1] = cos_peri*sin_incl;
  sat->matrix[0][2] = -cos_asc*sin_peri - sin_asc*cos_peri*cos_incl;
  sat->matrix[1][0] = -cos_asc*sin_incl;
  sat->matrix[1][1] = cos_incl;
  sat->matrix[1][2] = sin_asc*sin_incl;
  sat->matrix[2][0] = sin_asc*cos_peri + cos_asc*sin_peri*cos_incl;
  sat->matrix[2][1] = sin_peri*sin_incl;
  sat->matrix[2][2] = cos_asc*cos_peri - sin_asc*sin_peri*cos_incl;
  sat->matrix[0][3] = sat->matrix[1][3] = sat->matrix[2][3] = 0.0;
  sat->matrix[3][0] = sat->matrix[3][1] = sat->matrix[3][2] = 0.0;
  sat->matrix[3][3] = 1.0;
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*      smet_launch.c                                              */
/*      by Carlos I. Noriega                                       */
/*                                                                 */
/*      This file contains the routines used by SMET to            */
/*         launch a satellite.                                     */
/*                                                                 */
/*      Written - September 3, 1990                                */
/*      Modified - September 3, 1990                               */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

/*  ADD_LAUNCH                                          */
/*  Use a series of dialogs to get required info.       */
/*  See Chapter 3 for more information.                 */

add_launch()
{
  launch_struct *new_launch = (launch_struct *)malloc(sizeof(launch_struct));
  char str[15];

  dialog("Enter name of new satellite.", new_launch->name);
  new_launch->time = ask_time("time at burnout");
  dialog("Enter the altitude at burnout", str);
  new_launch->alt = atof(str)/distance_factor[unit_type];
  dialog("Enter the velocity at burnout.", str);
  new_launch->v = atof(str)/speed_factor[unit_type];
  dialog("Enter the latitude at burnout.", str);
  new_launch->lat = atof(str)*TO_RAD;
  dialog("Enter the longitude at burnout.", str);
  new_launch->lon = atof(str)*TO_RAD;
  dialog("Enter the angle above the horizon.", str);
  new_launch->elev = atof(str)*TO_RAD;
  dialog("Enter the heading angle.", str);
  new_launch->azim = atof(str)*TO_RAD;
  new_launch->next = pending_launches;
  pending_launches = new_launch;
}


/*  LAUNCH_PENDING_SATS                                 */
/*  Check all pending to see if it is time to launch.   */

launch_pending_sats()
{
  launch_struct *launch, *prev;

  for (launch = pending_launches; launch != NULL; launch = launch->next)
  {
    if (launch->time <= clock)
    {
      initialize_orbital_parameters(launch);
      if (launch = pending_launches) pending_launches = launch->next;
      else prev->next = launch->next;
      free((char *) launch);
    }
    prev = launch;
  }
}
```

```
/*  INITIALIZE_ORBITAL_PARAMETERS                          */
/*  See Chapter 3 for more information.                    */

initialize_orbital_parameters(launch)
launch_struct *launch;
{
  char str[70];
  float p,
  v_radial = launch->v * sin(launch->elev),
  v_ns = launch->v * cos(launch->elev) * cos(launch->azim),
  v_ew = launch->v * cos(launch->elev) * sin(launch->azim)
                                   + ROTATION_RATE * TU * cos(launch->lat),
  v_hor = sqrt(v_ns*v_ns + v_ew*v_ew);
  satellite *sat;

  make_new_current_sat(launch->name);
  sat = current_sat;

  /* adjust velocity, elevation angle and azimuth for earth spin  */
  launch->v = sqrt(v_radial*v_radial + v_hor*v_hor);
  launch->elev = atan2(v_radial,v_hor);
  launch->azim = atan2(v_ew,v_ns);

  /*  compute orbital parameters as discussed in Chapter 3 of thesis */
  sat->incl = acos(sin(launch->azim) * cos(launch->lat));
  sat->range = 1 + launch->alt;
  sat->major  = sat->range/(2 - sat->range * launch->v * launch->v);
  p = sat->range * v_hor * sat->range * v_hor;
  sat->ecc = fsqrt(1.0 - p/sat->major);
  sat->anom = atan2(p * tan(launch->elev), p - sat->range);
  sat->E = acos((sat->ecc + cos(sat->anom))/(1 + sat->ecc * cos(sat->anom)));
  if (sat->anom > DEG180) sat->E = DEG360 - sat->E;      /* get correct quadrant  */
  sat->epoch = launch->time - pow(sat->major,1.5) * (sat->E - sat->ecc*sin(sat->E)) * TU;
  sat->peri = fmod(atan2(sin(launch->lat),cos(launch->lat) * cos(launch->azim))
                                                  - sat->anom + DEG360, DEG360);
  sat->asc = fmod(launch->lon + clock*ROTATION_RATE - atan2(sin(launch->lat)
                           *sin(launch->azim), cos (launch->azim)) + DEG360, DEG360);

  update_static_parameters(sat);
  update_sat_matrix(sat);
  reset_dials();

  if (0.0 > sat->ecc || sat->ecc >= 1.0)
  {
    sprintf(str,"%s was launched into an open orbit and deleted from the system.",
            sat->name);
    delete_satellite();
  }
  else sprintf(str,"%s was launched.", sat->name);
  message(str);
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                    */
/*      smet_main.c                                                   */
/*      by Carlos I. Noriega                                          */
/*                                                                    */
/*      This file main routines used by SMET.                         */
/*                                                                    */
/*      Written - September 3, 1990                                   */
/*      Modified - September 3, 1990                                  */
/*                                                                    */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"


/*  MAIN                                                    */
/*  Initialize everything and enter a loop to check the     */
/*     queue, act on it and update the windows.             */

main(argc,argv)
int argc;
char **argv;
{
  initialize_smet(argc,argv);   /*  windows, menus, queue, variables     */
  while (TRUE)                  /* loop until we exit via a pop up menu  */
  {
    handle_queue();
    user_pre_update_satellites();  /* User defined routine  */
    update_satellites();
    user_post_update_satellites(); /* User defined routine  */
    draw_windows();
    if (clock_running) clock = clock + clock_speed; /* increment clock if on */
  }
}
```

```
/***************  This is where I want to do the multi processing  ***********/


/*  UPDATE_SATELLITES                                                  */
/*  Calling routine for updating all the satellites.                   */
/*      Also checks for earth impact and sets Automatic                */
/*      Viewing position if auto_viewer is on.                         */

update_satellites()
{
  satellite *sat, *previous_sat;
  float max_apogee = 1.2;

  previous_sat = &header_sat;
  launch_pending_sats();
  for(sat = header_sat.next; sat != NULL; sat = sat->next)
  {
    if (sat->maneuver != NULL && clock >= sat->maneuver->time) maneuver_sat(sat);
    user_pre_orbital_parameters(sat, clock);
    update_changing_parameters(sat, clock);
    user_post_orbital_parameters(sat, clock);
    if (sat->range <= 1.0) crash_satellite(sat, previous_sat);
    previous_sat = sat;
    max_apogee = max(sat->apogee_range, max_apogee);
  }

  if (auto_viewer)  /* put our viewing position out towards Ares */
  {
    viewer_x = max_apogee;
    viewer_y = 1.0;
    viewer_z = 1.0;
    viewer_range = sqrt(max_apogee*max_apogee + 2);
  }
}


/*  DRAW_WINDOWS                                                     */
/*  Calling routine for all the drawing routines.                    */

draw_windows()
{
  draw_omni_view();              /* draw the omniscient point of view   */
  draw_sat_view();               /* draw the satellite point of view    */
  draw_merc_view();              /* draw the mercator map with plot     */
  draw_parameters();             /* show the current satellite's stats */
  draw_sat_list();               /* draw the list of satellites         */
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                  */
/*      smet_maneuver.c                                             */
/*      by Carlos I. Noriega                                        */
/*                                                                  */
/*      This file contains the routines used by SMET to             */
/*         maneuver a satellite.                                    */
/*                                                                  */
/*      Written - September 3, 1990                                 */
/*      Modified - September 3, 1990                                */
/*                                                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"
/*  ADD_MANEUVER                                        */
/*  Use a series of dialogs to get required info.       */
/*  See Chapter 4 for more information.                 */

add_maneuver()
{
  char str[25];
  man_struct *walker, *maneuver;

  if (current_sat != NULL)
  {
    maneuver = (man_struct *)malloc(sizeof(man_struct));
    if (current_sat->maneuver == NULL)
    {
      dialog("Enter the True Anomoly at the time of the maneuver.", str);
      maneuver->anom = atof(str)*TO_RAD;
      current_sat->maneuver = maneuver;
      compute_maneuver_time(current_sat);
    }
    else
    {
      dialog("Enter the number of degrees to travel after the preceding maneuver.", str);
      maneuver->anom = atof(str)*TO_RAD;
      for (walker = current_sat->maneuver; walker->next != NULL; walker = walker->next) ;
      walker->next = maneuver;
    }
    dialog("Enter (1) for Delta-V/Yaw/Pitch or (2) Velocity/Flight Path/Heading Changes."
                                                                            , str);
    maneuver->normal_maneuver = atoi(str) % 2;
    if (maneuver->normal_maneuver)
    {
      dialog("Enter the Delta-V for the maneuver.", str);
      maneuver->deltav = atof(str)/speed_factor[unit_type];
      dialog("Enter the Pitch for the maneuver.", str);
      maneuver->pitch = atof(str)*TO_RAD;
      dialog("Enter the Yaw for the maneuver.", str);
      maneuver->yaw = atof(str)*TO_RAD;
    }
    else
    {
      dialog("Enter the Velocity Magnitude Change for the maneuver.", str);
      maneuver->deltav = atof(str)/speed_factor[unit_type];
      dialog("Enter the Flight Path Change for the maneuver.", str);
      maneuver->pitch = atof(str)*TO_RAD;
      dialog("Enter the Heading Change for the maneuver.", str);
      maneuver->yaw = atof(str)*TO_RAD;
    }
    maneuver->next = NULL;
  }
}
```

```
/*  MANEUVER_SAT                                                    */
/*  See Chapter 4 for more information.                             */

maneuver_sat(sat)
satellite *sat;
{
  float delta_f_path, delta_heading, new_velocity,
        old_ecc = sat->ecc,
        old_major = sat->major,
        old_incl = sat->incl,
        old_peri = sat->peri,
        old_asc = sat->asc;

  update_changing_parameters(sat, sat->maneuver->time);

  if (sat->maneuver->normal_maneuver)
  {
    compute_deltas(sat, &delta_f_path, &delta_heading, &new_velocity);
    flight_path_change(sat, delta_f_path);
    heading_change(sat, delta_heading);
    velocity_change(sat, new_velocity);
  }
  else
  {
    flight_path_change(sat, sat->maneuver->pitch);
    heading_change(sat, sat->maneuver->yaw);
    velocity_change(sat, sat->velocity + sat->maneuver->deltav);
  }

  if (0.0 <= sat->ecc && sat->ecc < 1.0)
  {
    sat->epoch = sat->maneuver->time
                         - pow(sat->major,1.5)*(sat->E - sat->ecc * sin(sat->E)) * TU;
    if (sat = current_sat) reset_dials();
    update_static_parameters(sat);
    update_sat_matrix(sat);
  }
  else
  {
    message("That maneuver put the satellite into an open orbit and will be ignored.");
    sat->ecc = old_ecc;
    sat->major = old_major;
    sat->incl = old_incl;
    sat->peri = old_peri;
    sat->asc = old_asc;
  }
  delete_maneuver(sat);
}
```

```
/*  COMPUTE_DELTAS                                          */
/*  See Chapter 4 for more information.                     */

compute_deltas(sat, delta_f_path, delta_heading, new_velocity)
satellite *sat;
float *delta_f_path, *delta_heading, *new_velocity;
{
  float
    delvp = sat->maneuver->deltav * cos(sat->maneuver->yaw),
    delvps = delvp * sin(sat->maneuver->pitch),
    delvop = sat->maneuver->deltav * sin(sat->maneuver->yaw),
    vlmodl = sat->velocity + delvp * cos(sat->maneuver->pitch),
    vlmod2 = fsqrt(delvps*delvps + vlmodl*vlmodl);

  *delta_f_path = fatan2(delvps,vlmodl);
  *delta_heading = fatan2(delvop, vlmod2);
  *new_velocity = fsqrt(delvop*delvop + vlmod2*vlmod2);
}

/*  COMPUTE_MANEUVER_TIME                                   */
/*  Set the time of maneuver based on the current time,     */
/*  current anomaly and desired anomaly.                    */

compute_maneuver_time(sat)
satellite *sat;
{
  float c_anom = cos(sat->maneuver->anom),
        E = acos((sat->ecc + c_anom)/(1 + sat->ecc * c_anom));  /* Eccentric Anomoly */
        /* number of revolutions completed plus number of more whole revolutions */
  int revs = (int)(clock - sat->epoch)/sat->period
               + (int)(sat->maneuver->anom/DEG360);

  /* go to next time around if we're past it on this revolution*/
  if (sat->maneuver->anom <= sat->anom) revs++;

  /* make anomoly positive  */
  while (sat->maneuver->anom < 0) sat->maneuver->anom = sat->maneuver->anom + DEG360;

  /* correct quadrant  */
  if (fmod(sat->maneuver->anom,DEG360) > DEG180) E = DEG360 - E;

  sat->maneuver->time = sat->epoch + sat->period*(revs + (E - sat->ecc * sin(E))/DEG360);
}

/*  DELETE_MANEUVER                                         */
/*  Remove the maneuver from the linked list.               */

delete_maneuver(sat)
satellite *sat;
{
  man_struct *temp;

  temp = sat->maneuver;
  sat->maneuver = sat->maneuver->next;
  free((char *) temp);

  /*  set execution time of next maneuver if there is one */
  if (sat->maneuver != NULL)
  {
    sat->maneuver->anom = sat->maneuver->anom + sat->anom;
    compute_maneuver_time(sat);
  }
}
```

```
/*  FLIGHT_PATH_CHANGE                                        */
/*  See Chapter 4 for information.                            */

flight_path_change(sat, delta)   /* In-Plane Maneuver */
satellite *sat;
float delta;
{
  float p,
        old_anom,
        tan_f_path;
  char str[70];

  if (nonzerop(delta))
  {
    tan_f_path = sat->ecc * sin(sat->E)/fsqrt( 1 - sat->ecc * sat->ecc);
    sat->ecc = fsqrt(1 - (1 - sat->ecc*sat->ecc) *
                            pow((cos(delta) - tan_f_path*sin(delta)),2));
    old_anom = sat->anom;
    tan_f_path = (tan_f_path + tan(delta))/( 1 - tan_f_path * tan(delta));
    p = sat->major * (1 - sat->ecc * sat->ecc);
    sat->anom = fmod(fatan2(p * tan_f_path,p - sat->range) + DEG360, DEG360);
    sat->E = acos((sat->ecc + cos(sat->anom))/(1 + sat->ecc * cos(sat->anom)));
    if (sat->anom > DEG180) sat->E = DEG360 - sat->E;      /* get correct quadrant  */

    sat->peri = fmod(sat->peri + old_anom - sat->anom + DEG360, DEG360);
    sprintf(str,"%s executed a flight path change of %4.3f degrees.",
                  sat->name, delta * TO_DEG);
    message(str);
  }
}
```

```
/*  HEADING_CHANGE                                           */
/*  See Chapter 4 for information.                           */

heading_change(sat, delta)  /* Out-of-Plane Maneuver */
satellite *sat;
float delta;
{
  float incl_old = sat->incl,
        u_old = sat->peri + sat->anom,
        u_new = 0.0,
        temp;
  char str[70];

  if (nonzerop(delta))
  {
    sat->incl = facos(cos(incl_old)*cos(delta) + sin(incl_old)*cos(u_old)*sin(delta));
    if (nonzerop(sat->incl))
    {
      temp = sin(incl_old)*cos(u_old)*cos(delta) - cos(incl_old)*sin(delta);
      u_new = facos(temp/sin(sat->incl));
      if (sat->y < 0.0) u_new = DEG360 - u_new;    /* get correct quadrant  */
    }
    sat->peri = fmod(u_new - sat->anom + DEG360, DEG360);
    sat->asc = fmod(sat->asc + fatan2(sin(u_old)*cos(incl_old),cos(u_old))
                   - fatan2(sin(u_new)*cos(sat->incl), cos(u_new)) + DEG360, DEG360);
    sprintf(str,"%s executed a heading change of %4.3f degrees.",sat->name,delta*TO_DEG);
    message(str);
  }
}
```

```
/*  VELOCITY_CHANGE                                                  */
/*  See Chapter 4 for information.                                   */

velocity_change(sat, new_vel)
satellite *sat;
float new_vel;
{
  float tan_f_path, cos_f_path, p, old_anom;
  char str[70];

  if (nonzerop(new_vel - sat->velocity))
  {
    if (sat->ecc == 0)
      if (new_vel > sat->velocity)
      {
        sat->peri = fmod(sat->peri + sat->anom, DEG360);
        sat->E = sat->anom = 0.0;
      }
      else
      {
        sat->peri = fmod(sat->peri + sat->anom + DEG180, DEG360);
        sat->E = sat->anom = DEG180;
      }
    sat->major  = sat->range/(2 - sat->range * new_vel * new_vel);

    tan_f_path = sat->ecc * fsin(sat->E)/sqrt(1 - sat->ecc * sat->ecc);
    cos_f_path = cos(atan(tan_f_path));
    p = sat->range * sat->range * new_vel * new_vel * cos_f_path * cos_f_path;
    sat->ecc = fsqrt(1.0 - p/sat->major);

    old_anom = sat->anom;
    sat->anom = fmod(fatan2(p * tan_f_path,p - sat->range) + DEG360, DEG360);
    sat->E = acos((sat->ecc + cos(sat->anom))/(1 + sat->ecc * cos(sat->anom)));
    if (sat->anom > DEG180) sat->E = DEG360 - sat->E;       /* get correct quadrant  */

    sat->peri = fmod(sat->peri + old_anom - sat->anom + DEG360, DEG360);

    sprintf(str,"%s executed a velocity magnitude change of %4.3f %s.",sat->name,
               (new_vel - sat->velocity)*speed_factor[unit_type],speed_unit[unit_type]);
    message(str);
  }
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                  */
/*      smet_map.c                                                  */
/*      by Carlos I. Noriega                                        */
/*                                                                  */
/*      This file contains the routines to draw the globe           */
/*         used by the SMET demonstrator.                           */
/*                                                                  */
/*      Written - September 3, 1990                                 */
/*      Modified - September 3, 1990                                */
/*                                                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

/*   DRAW-MERCATOR                                        */
/*                                                        */

draw_mercator()
{
  c3f(Color[EARTH]);
  rectf(-DEG180,-DEG90,DEG180,DEG90); /* map background */

  draw_coasts2d();
  draw_borders2d();
  draw_latlons2d();
}




/*   DRAW-COASTS2D                                        */
/*   Draw the desired data FINE or COURSE.                */

draw_coasts2d()
{
  int c;

  linewidth(2);
  c3f(Color[COAST]);
  if (fineflag)
    for (c = 0; c < FINEPOINTS; c++)
      if (fine2d_data.pen[c])
        draw(fine2d_data.x[c], fine2d_data.y[c], fine2d_data.z[c]);
      else
        move(fine2d_data.x[c], fine2d_data.y[c], fine2d_data.z[c]);
  else
    for (c = 0; c < COARSEPOINTS; c++)
      if (coarse2d_data.pen[c])
        draw(coarse2d_data.x[c], coarse2d_data.y[c], coarse2d_data.z[c]);
      else
        move(coarse2d_data.x[c], coarse2d_data.y[c], coarse2d_data.z[c]);
}
```

```
/*  DRAW-BORDERS2D                                          */
/*  Draw the border data if desired.                        */

draw_borders2d()
{
  int c;

  linewidth(1);
  c3f(Color[BORDER]);
  if (borderflag)
    for (c = 0; c < BORDERPOINTS; c++)
      if (border2d_data.pen[c])
        draw(border2d_data.x[c], border2d_data.y[c], border2d_data.z[c]);
      else
        move(border2d_data.x[c], border2d_data.y[c], border2d_data.z[c]);
}




/*  DRAW-LATLONS2D                                          */
/*  Draw the latitude and longitude lines on the map.       */

draw_latlons2d()
{
  int c;

  linewidth(1);
  c3f(Color[LATLONG]);
  for (c = -6; c <= 6; c++)   /*  draw the longitudes, every 30 degrees      */
  {
    move(DEG30 * c,-DEG90, 0.0);
    draw(DEG30 * c, DEG90, 0.0);
  }
  for (c = -3; c <= 3; c++)   /*  draw the latitudes, every 30 degrees       */
  {
    move(-DEG180, DEG30 * c, 0.0);
    draw( DEG180, DEG30 * c, 0.0);
  }
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*      smet_orbit.c                                                 */
/*      by Carlos I. Noriega                                         */
/*                                                                   */
/*      This file contains the routine used by SMET to               */
/*         draw the orbit ellipses in a viewing window.              */
/*                                                                   */
/*      Written - September 3, 1990                                  */
/*      Modified - September 3, 1990                                 */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

/*  DRAW-ORBIT                                          */
/*  draw an orbital ellipse based on the global         */
/*      parameters: major, minor and eccentricity.      */

draw_orbit(sat)
satellite *sat;
{
  pushmatrix();
  multmatrix(sat->matrix);  /* set the orbital transformation plane matrix */

  linewidth(3);                /*  set the linewidth */
  c4f(Color[sat->color]);      /*  go to satellite's color    */

  if (sat->vectors)
  { /* draw the normal vector  */
    move(0.0, 1.0, 0.0);
    draw(0.0, 5.0, 0.0);

    /* draw the perigee vector  */
    move(0.0, 0.0, 1.0);
    draw(0.0, 0.0, sat->perigee_range);
  }

  /*  shift the focus to the origin     */
  translate(0.0, 0.0, -sat->ecc*sat->major);

  /*  make an ellipse out of a circle   */
  scale(sat->minor, 1.0, sat->major);

  /*  put it on equatorial plane        */
  rotate(900,'x');

  if (sat->fill)
    {                 /*  draw a simple transparent circle  */
    zbuffer(FALSE);
    blendfunction(BF_SA, BF_MSA);
    circf(0.0,0.0,1.0);
    blendfunction(BF_ONE, BF_ZERO);
    zbuffer(TRUE);
  }
  else circ(0.0,0.0,1.0);     /*  draw an unfilled oval   */

  popmatrix();
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*      smet_param.c                                               */
/*      by Carlos I. Noriega                                       */
/*                                                                 */
/*      This file contains the routines used by SMET to            */
/*         handle parameter changes.                               */
/*                                                                 */
/*      Written - September 3, 1990                                */
/*      Modified - September 3, 1990                               */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"


smet_param_option(n)     /* used by popup menu */
int n;
{
  param_option = n;
}



smet_sat_color(n)        /* used by popup menu */
int n;
{
  current_sat->color = SAT_RED + n;
}


/*  CHANGE_PARAMETER                                      */
/*  Compute the parameter number base on the mouse        */
/*  position and call the appropriate change routine.     */

change_parameter(y)
int y;
{
  int paramnum = (float)(satlistwin.sizey - y) / (float)LINE - 4;

  if (0 <= paramnum && paramnum < 25)
    switch (param_option) {
      case CLASSICAL:   change_classical_param(paramnum); break;
      case SECONDARY:   change_secondary_param(paramnum); break;
      case POSITIONS:   change_position_param(paramnum); break;
      case OPTIONS:     change_options_param(paramnum); break;
      case COMBINED:    change_combined_param(paramnum); break;
    }
  else change_sysparam(paramnum - 29);
}
```

```
/*  CHANGE_xxxxxxxx_PARAM                              */
/*  Change the appropriate parameter.                  */

change_classical_param(paramnum)
int paramnum;
{
  switch(paramnum)
  {
    case 0:  ask_name(); break;
    case 1:  ask_major(); update_static_parameters(current_sat); break;
    case 2:  ask_minor_for_ecc(); update_static_parameters(current_sat); break;
    case 3:  ask_ecc(); update_static_parameters(current_sat); break;
    case 4:  ask_incl(); update_sat_matrix(current_sat);
             update_perturb_constants(current_sat); break;
    case 5:  ask_asc(); update_sat_matrix(current_sat); break;
    case 6:  ask_peri(); update_sat_matrix(current_sat); break;
    case 7:  bad_choice_message(classicalparamblock.params[paramnum].name); break;
    case 8:  ask_epoch(); delete_plots(current_sat); break;
  }
  if (1 <= paramnum && paramnum <= 6)
  {
    reset_dials();
    delete_plots(current_sat);
  }
}


change_secondary_param(paramnum)
int paramnum;
{
  if (paramnum == 0) ask_name();
    else if (paramnum == 3) ask_epoch();
      else if (paramnum <= 6)
        bad_choice_message(secondaryparamblock.params[paramnum].name);
}


change_position_param(paramnum)
int paramnum;
{
  if (paramnum == 0) ask_name();
    else if (paramnum <= 7)
      bad_choice_message(positionparamblock.params[paramnum].name);
}


change_options_param(paramnum)
int paramnum;
{
  char str[50];

  switch(paramnum)
  {
    case 0: ask_name(); break;
    case 1: dopup(color_menu); break;
    case 2: current_sat->fill = !current_sat->fill; break;
    case 3: current_sat->vectors = !current_sat->vectors; break;
    case 4: current_sat->plot = !current_sat->plot; delete_plots(current_sat);
  }
}
```

```
change_combined_param(paramnum)
int paramnum;
{
  if (paramnum <= 24) switch(paramnum)
  {
    case 0:  ask_name(); break;
    case 1:  ask_major(); update_static_parameters(current_sat); break;
    case 2:  ask_minor_for_ecc(); update_static_parameters(current_sat); break;
    case 3:  ask_ecc(); update_static_parameters(current_sat); break;
    case 4:  ask_incl(); update_sat_matrix(current_sat);
             update_perturb_constants(current_sat); break;
    case 5:  ask_asc(); update_sat_matrix(current_sat); break;
    case 6:  ask_peri(); update_sat_matrix(current_sat); break;
    case 8:  ask_epoch(); delete_plots(current_sat); break;
    case 21: dopup(color_menu); break;
    case 22: current_sat->fill = !current_sat->fill; break;
    case 23: current_sat->vectors = !current_sat->vectors; break;
    case 24: current_sat->plot = !current_sat->plot; delete_plots(current_sat); break;
    default: bad_choice_message(combinedparamblock.params[paramnum].name); break;
  }
  if (1 <= paramnum && paramnum <= 6)
  {
    reset_dials();
    delete_plots(current_sat);
  }
}
```

```
change_sysparam(paramnum)
int paramnum;
{
  char str[15];
  satellite *sat;

  switch(paramnum)
  {
    case 0: clock_running = !clock_running; break;
    case 1:
        clock = ask_time("current time");
        for (sat = current_sat; sat != NULL; sat = sat->next) delete_plots(sat);
        break;
    case 2:
        dialog("Enter a clock speed (in minutes per update)",str);
        clock_speed = atof(str);
        setvaluator(DIAL3, (short)(clock_speed), 0, max((short)(clock_speed)*1.2, 360));
        break;
    case 3: fineflag = !fineflag; break;
    case 4: borderflag = !borderflag; break;
    case 5: perturbations = !perturbations; break;
    case 6: auto_viewer = !auto_viewer; break;
    case 7:
        dialog("Enter new viewer X position.",str);
        viewer_x = atof(str)/distance_factor[unit_type];
        viewer_range = sqrt(viewer_x*viewer_x + viewer_y*viewer_y + viewer_z*viewer_z);
        auto_viewer = FALSE;
        break;
    case 8:
        dialog("Enter new viewer Y position.",str);
        viewer_y = atof(str)/distance_factor[unit_type];
        viewer_range = sqrt(viewer_x*viewer_x + viewer_y*viewer_y + viewer_z*viewer_z);
        auto_viewer = FALSE;
        break;
    case 9:
        dialog("Enter new viewer Z position.",str);
        viewer_z = atof(str)/distance_factor[unit_type];
        viewer_range = sqrt(viewer_x*viewer_x + viewer_y*viewer_y + viewer_z*viewer_z);
        auto_viewer = FALSE;
        break;
    case 10: unit_type = (unit_type + 1) % 3;
  }
}



/*  BAD_CHOICE_MESSAGE                                          */
/*                                                              */

bad_choice_message(name)
char *name;
{
  char str[50];

  sprintf(str,"%scannot be changed by the user.",name);
  message(str);
}
```

```
/*  ASK_xxxxxx  -  Custom dialog routines used          */
/*   to ensure the correct input is made.               */
float ask_time(name)
char *name;
{
  char  msg[75],
        response[20];
  int   hour = 0,
        min = 0;
  float sec = 0.0,
        time;

  sprintf(msg, "Enter the %s.",name);
  dialog(msg, response);
  sscanf(response,"%d%*c%d%*c%f",&hour, &min, &sec);
  if (min >= 0 && min < 60 && sec >= 0.0 && sec < 60.0)
    if (hour >= 0 && response[0] != '-') time = hour*60 + min + sec/60;
    else time = hour*60 - min - sec/60;
  else
  {
    sprintf(msg, "The %s must be entered in 00:00:00 format.", name);
    message(msg);
    time = ask_time(name);
  }
  return(time);
}


ask_name()
{
  char str[45];

  sprintf(str, "Enter new name for satellite %s -", current_sat->name);
  dialog(str,current_sat->name);
}


ask_major()
{
  char str[15];

  dialog("Enter a Semi-Major Axis.",str);
  current_sat->major = atof(str)/distance_factor[unit_type];
  if (current_sat->major <= 0.0)
  {
    message("Semi-Major Axis must greater than 0.");
    ask_major();
  }
}


ask_minor_for_ecc()
{
  char str[15];

  dialog("Enter a Semi-Minor Axis.",str);
  current_sat->minor = atof(str)/distance_factor[unit_type];
  current_sat->ecc = sqrt( 1 - pow(current_sat->minor / current_sat->major, 2));
  if (current_sat->minor <= 0.0 || current_sat->minor > current_sat->major)
  {
    message("Semi-Minor Axis must be between 0 and the Semi-Major Axis.");
    ask_minor_for_ecc();
  }
}
```

```
ask_ecc()
{
  char str[15];

  dialog("Enter an Eccentricity.",str);
  current_sat->ecc = atof(str);
  if (current_sat->ecc < 0.0 || current_sat->ecc >= 1.0)
  {
    message("Eccentricity must be between 0.0 and 1.0.");
    ask_ecc();
  }
}


ask_incl()
{
  char str[15];

  dialog("Enter an Inclination.",str);
  current_sat->incl = atof(str) * TO_RAD;
  if (current_sat->incl < 0.0 || current_sat->incl > DEG180)
  {
    message("Inclination must be between 0 and 180 degrees.");
    ask_incl();
  }
}


ask_asc()
{
  char str[15];

  dialog("Enter an Ascending Node.",str);
  current_sat->asc = atof(str) * TO_RAD;
  if (current_sat->asc < 0.0 || current_sat->asc > DEG360)
  {
    message("Ascending Node must be between 0 and 360 degrees.");
    ask_asc();
  }
}


ask_peri()
{
  char str[15];

  dialog("Enter an Argument of Periapsis.",str);
  current_sat->peri = atof(str) * TO_RAD;
  if (current_sat->peri < 0.0 || current_sat->peri > DEG360)
  {
    message("Argument of Periapsis must be between 0 and 360 degrees.");
    ask_peri();
  }
}

ask_epoch()
{
  current_sat->epoch = ask_time("epoch");
}
```

```c
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                  */
/*      smet_queue.c                                                */
/*      by Carlos I. Noriega                                        */
/*                                                                  */
/*      This file contains the routines used by SMET to             */
/*         check the queue and handle events not covered            */
/*         in other files.                                          */
/*                                                                  */
/*      Written - September 3, 1990                                 */
/*      Modified - September 3, 1990                                */
/*                                                                  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

/*  GET_DIAL_VALUE                                            */
/*  Given a dial and an initial reading from the queue keep  */
/*     reading it until the user stops tuning it.             */

float get_dial_value(dev,initial_value)
Device dev;            /*  which dial we are working with          */
short initial_value;   /*  value when first detected on the queue  */
{
  /*  set the current value of the dial            */
  short value = initial_value;

  /*  read the queue until something other than the */
  /*   same dial is on top  or the queue is empty   */
  while (qtest() == dev) qread(&value);

  if (dev != DIAL3) delete_plots(current_sat);

  /*  return the last value in float format         */
  return((float)value);
}


/*  RESET_DIALS                                              */
/*  Used when a value of a classical parameter is changed   */
/*  without the use of the dials.  Keeps everyone happy.    */

reset_dials()
{
  /*  Set the dials allowing for degree of accuracy desired */
  setvaluator(DIAL2, (short)(current_sat->peri*TO_DEG*10), 0, 360*10);
  setvaluator(DIAL4, (short)(current_sat->incl*TO_DEG*10), 0, 180*10);
  setvaluator(DIAL5, (short)(current_sat->asc*TO_DEG*10), 0, 360*10);
  setvaluator(DIAL6, (short)(current_sat->major*100), 100, 1000);
  setvaluator(DIAL7, (short)(current_sat->ecc*1000), 0, 999);
  qreset();  /*  suppress "setvaluator" induced events. */
}
```

```
/*  HANDLE_QUEUE                                              */
/*  Clears the queue of all pending events.                   */

handle_queue()
{
  short value;        /*  value from a device on the queue    */

  while (qtest())     /*  clear queue before updating windows */
    switch(qread(&value))
      {
      case MIDDLEMOUSE:
        if (value == 1 ) handle_middlemouse();
        break;
      case RIGHTMOUSE:
        if (value == 1 ) dopup(top_menu);
        break;
      case DIAL2:             /*   Adjust "peri"              */
        if (current_sat != NULL)
        {
          current_sat->peri = get_dial_value(DIAL2,value)*TO_RAD/10;
          update_sat_matrix(current_sat);
        }
        break;
      case DIAL3:             /*   Adjust "clock speed"       */
        clock_speed = get_dial_value(DIAL3,value);
        break;
      case DIAL4:             /*   Adjust "incl"              */
        if (current_sat != NULL)
        {
          current_sat->incl = get_dial_value(DIAL4,value)*TO_RAD/10;
          update_sat_matrix(current_sat);
          update_perturb_constants(current_sat);
        }
        break;
      case DIAL5:             /*   Adjust "asc"               */
        if (current_sat != NULL)
        {
          current_sat->asc = get_dial_value(DIAL5,value)*TO_RAD/10;
          update_sat_matrix(current_sat);
        }
        break;
      case DIAL6:             /*   Adjust "major"             */
        if (current_sat != NULL)
        {
          current_sat->major = get_dial_value(DIAL6,value)/100;
          update_static_parameters(current_sat);
        }
        break;
      case DIAL7:             /*   Adjust "ecc"               */
        if (current_sat != NULL)
        {
          current_sat->ecc = get_dial_value(DIAL7,value)/1000;
          update_static_parameters(current_sat);
        }
        break;
      default: break;
      }  /* end of switch statement */
}
```

```
/*  PROGRAM-EXIT                              */
/*  Exit the program cleanly                  */

program_exit()
{
  char str[10];
  dialog("Would you like to save any satellites before quitting?",str);
  if (str[0] == 'y' || str[0] == 'Y') ask_and_write_sat_file();
  message("Thank you for using SMET.");
  winclose(satlistwin.gid);
  winclose(paramwin.gid);
  winclose(auxwin1.gid);
  winclose(auxwin2.gid);
  winclose(mainwin.gid);
  winclose(dialogwin.gid);
  winclose(backwin.gid);
  gexit();  /*  Close graphics ports  */
  exit();   /*  Exit orbital_demo     */
}



/*  HANDLE-MIDDLEMOUSE                              */
/*  Call the appropriate routine based on the      */
/*  mouse position.                                 */

handle_middlemouse()
{
  short x,y;  /*   mouse position       */

  qread(&x);  /*  get MOUSEX            */
  qread(&y);  /*  get MOUSEY            */

  /*  Are we in the area containing the aux windows?          */
  if ( x >=  auxwin1.orgx && x <= auxwin2.orgx+auxwin2.sizex)
    {
      if ( y >=  auxwin1.orgy && y <= auxwin1.orgy+auxwin1.sizey )
        switch_windows(x);
    }
  else if ( x <= paramwin.orgx + paramwin.sizex) change_parameter(y);
       else if ( x >= satlistwin.orgx) switch_current_sat(y);
}
```

```
/*  SWITCH-WINDOWS                                                       */
/*  See if we are in one of the auxiliaty windows.  If we are, */
/*      then swap the gid values for the views associated with */
/*      the current main window and the auxiliary window which */
/*      was selected by the user by pressing the MIDDLEMOUSE.  */
/*      The window_order variable designates the which views   */
/*      are in which windows.  For example, OMNISATMERC        */
/*      means that the main window contains the Omniscient     */
/*      point of view, auxiliary window 1 contains the         */
/*      Satellite point of view, and auxiliary window 2        */
/*      contains the meracator projection.                     */

switch_windows(x)
short x;
{
  /*  Are we in the left aux window?                      */
  if ( x <=  auxwin1.orgx + auxwin1.sizex )
    switch (window_order){
    case OMNISATMERC:
      sat_view_gid = mainwin.gid;
      omni_view_gid = auxwin1.gid;
      window_order = SATOMNIMERC;
      break;
    case OMNIMERCSAT:
      merc_view_gid = mainwin.gid;
      omni_view_gid = auxwin1.gid;
      window_order = MERCOMNISAT;
      break;
    case SATMERCOMNI:
      merc_view_gid = mainwin.gid;
      sat_view_gid = auxwin1.gid;
      window_order = MERCSATOMNI;
      break;
    case SATOMNIMERC:
      omni_view_gid = mainwin.gid;
      sat_view_gid = auxwin1.gid;
      window_order = OMNISATMERC;
      break;
    case MERCOMNISAT:
      omni_view_gid = mainwin.gid;
      merc_view_gid = auxwin1.gid;
      window_order = OMNIMERCSAT;
      break;
    case MERCSATOMNI:
      sat_view_gid = mainwin.gid;
      merc_view_gid = auxwin1.gid;
      window_order = SATMERCOMNI;
      break;
    }
```

```
/*  Are we in the right aux window?                                   */
else if (x >= auxwin2.orgx )
  switch (window_order){
  case OMNISATMERC:
    merc_view_gid = mainwin.gid;
    omni_view_gid = auxwin2.gid;
    window_order = MERCSATOMNI;
    break;
  case OMNIMERCSAT:
    sat_view_gid = mainwin.gid;
    omni_view_gid = auxwin2.gid;
    window_order = SATMERCOMNI;
    break;
  case SATMERCOMNI:
    omni_view_gid = mainwin.gid;
    sat_view_gid = auxwin2.gid;
    window_order = OMNIMERCSAT;
    break;
  case SATOMNIMERC:
    merc_view_gid = mainwin.gid;
    sat_view_gid = auxwin2.gid;
    window_order = MERCOMNISAT;
    break;
  case MERCOMNISAT:
    sat_view_gid = mainwin.gid;
    merc_view_gid = auxwin2.gid;
    window_order = SATOMNIMERC;
    break;
  case MERCSATOMNI:
    omni_view_gid = mainwin.gid;
    merc_view_gid = auxwin2.gid;
    window_order = OMNISATMERC;
    break;
  }
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                     */
/*      smet_sat.c                                                     */
/*      by Carlos I. Noriega                                           */
/*                                                                     */
/*      This file contains the routines used by SMET to                */
/*      draw the satellite in a viewing window.  It was borrowed       */
/*      from the ASAT program written by Jim Zanoli and Do Kyeing      */
/*      Ok.  It was modified and converted to procedures.              */
/*                                                                     */
/*      Written - September 3, 1990                                    */
/*      Modified - September 3, 1990                                   */
/*                                                                     */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"


/*  DRAW-SATELLITE                                          */
/*  draws a satellite in the correct attitute in            */
/*      the appropriate spot in the orbital plane           */
/*      as set by the global orbital parameters             */
/*      anom, r_p and r_q. The satellite is scaled          */
/*      with respect to the major axis of the orbit.*/

draw_satellite(sat)
satellite *sat;
{       /* scale size increases with distance     */
  float sat_scale = 0.005 * viewer_range;  /* maybe vr*vr  ??  */
  short angle;   /*  dummy loop counter for making the ball         */

  pushmatrix();
  multmatrix(sat->matrix);  /* set the orbital transformation plane matrix */
  linewidth(2);                 /*  set the linewidth                 */

  translate(sat->q,0.0,sat->p);      /*  move it in the orbital plane      */
  rot(sat->anom*TO_DEG,'y');               /*  point towards the earth          */
  rotate(900,'x');              /*  lay it on its side                */
  scale(sat_scale,sat_scale,sat_scale);   /*  make much smaller than the earth  */

  c3f(Color[WHITE]);     /*  go white   */

  /* draw left wing */
  pushmatrix();
  scale(3.0,0.7,0.1);         /*  make it long, flat, and thin  */
  translate(-1.2,0.0,0.0);  /*  move it out to the side       */
  draw_cube();
  popmatrix();

  /* draw right wing */
  pushmatrix();
  scale(3.0,0.7,0.1);         /*  make it long, flat, and thin  */
  translate(1.2,0.0,0.0);    /*  move it out to the side       */
  draw_cube();
  popmatrix();

  c3f(Color[sat->color]);         /*  make the body the right color      */
```

```
    /* draw satellite body */
    draw_cylinder();

    /* draw satellite middle globe part */
    pushmatrix();
    translate(0.0,-3.5,0.0);   /* move below the main cylinder  */

    c3f(Color[YELLOW]);   /* go yellow  */

    /*  draw a 0.8 radius ball          */
    for(angle=0;angle<=3600;angle=angle+200)
      {
        rotate(300,'x');
        circf(0.0,0.0,0.8);
        rotate(300,'y');
        circf(0.0,0.0,0.8);
        rotate(300,'z');
        circf(0.0,0.0,0.8);
      }
    popmatrix();

    c3f(Color[sat->color]);        /*  make the body the right color      */

    /* draw front small body */
    pushmatrix();
    translate(0.0,-5.0,0.0);  /* move below the ball   */
    scale(1.0,0.3,1.0);       /*  make it shorter      */
    draw_cylinder();
    popmatrix();

    popmatrix();
  }
```

```
/*  DRAW-CYLINDER                                       */
/*  draw a 6 unit long by 1 unit radius cylinder        */
/*      centered at the origin and sitting upright      */

draw_cylinder()
{
  short angle;

  /*  draw the sides of the cylinder       */
  /*  by drawing 72 filled rectangles      */
  pushmatrix();
  for(angle=0;angle<=3600;angle=angle+50)
    {
      rotate(50,'y');
      rectfi(-1,-3,1,3);
    }
  popmatrix();

  /*  draw the top of the cylinder         */
  pushmatrix();
  translate(0.0,3.0,0.0);  /*  move it to the top        */
  rotate(900,'x');         /*  lay it flat    */
  circfi(0,0,1);
  popmatrix();

  /*  draw the bottom of the cylinder      */
  pushmatrix();
  translate(0.0,-3.0,0.0);  /* move it to the bottom    */
  rotate(-900,'x');         /*  lay it flat    */
  circfi(0,0,1);
  popmatrix();
}
```

```
/*  DRAW-CUBE                                          */
/*  draw a cube, used for reshaping to any box shape */

draw_cube()
{
  /* draw the front */
  pushmatrix();
  translate(0.0,0.0,1.0);
  rectfi(-1,-1,1,1);
  popmatrix();

  /* draw the back */
  pushmatrix();
  translate(1.0,0.0,0.0);
  rotate(900,'y');
  rectfi(-1,-1,1,1);
  popmatrix();

  /* draw the right */
  pushmatrix();
  translate(0.0,0.0,-1.0);
  rotate(1800,'y');
  rectfi(-1,-1,1,1);
  popmatrix();

  /* draw the left */
  pushmatrix();
  translate(-1.0,0.0,0.0);
  rotate(-900,'y');
  rectfi(-1,-1,1,1);
  popmatrix();

  /* draw the top */
  pushmatrix();
  translate(0.0,1.0,0.0);
  rotate(-900,'x');
  rectfi(-1,-1,1,1);
  popmatrix();

  /* draw the bottom */
  pushmatrix();
  translate(0.0,-1.0,0.0);
  rotate(900,'x');
  rectfi(-1,-1,1,1);
  popmatrix();
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*      smet_user.c                                                  */
/*      by Carlos I. Noriega                                         */
/*                                                                   */
/*      This file contains the user routines called by SMET.         */
/*      They currently do nothing.                                   */
/*                                                                   */
/*      Written - September 3, 1990                                  */
/*      Modified - September 3, 1990                                 */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"

user_init()
{

/* ex. read sat files, init variables etc.  */

}


user_new_sat()
{

/* ex. modify new current_sat  */

}


user_pre_update_satellites()
{

/* ex. check for exit, compare sat positions, maneuver sats, launch sats, etc.  */

}

user_post_update_satellites()
{

/* ex. check for exit, compare sat positions, maneuver sats, launch sats, etc.  */

}


user_pre_orbital_parameters(sat, time)
satellite *sat;
float time;
{

/* ex. perturb orbit etc.  */

}

user_post_orbital_parameters(sat, time)
satellite *sat;
float time;
{

/* ex. perturb orbit etc.  */
```



```
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                   */
/*      smet_windows.c                                               */
/*      by Carlos I. Noriega                                         */
/*                                                                   */
/*      This file contains the routines used by SMET to              */
/*         draw all the windows.                                     */
/*                                                                   */
/*      Written - September 3, 1990                                  */
/*      Modified - September 3, 1990                                 */
/*                                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "smet.h"



/*  DRAW-OMNI-VIEW                                          */
/*  Draw the globe, orbital ellipse, satellite and          */
/*      and the IJK axis at the center of the window        */
/*      form an omniscient point of view.                   */

draw_omni_view()
{
  satellite     *sat;

  /*  select the proper window and and clear it                 */
  winset(omni_view_gid);
  c3f(Color[BLACK]);
  zclear();
  clear();

  /*  make the box large enough to hold any possible rotation */
  /*  of the orbital plane.  Keep the globe in the center.    */
  ortho(-viewer_range, viewer_range, -viewer_range, viewer_range, 0, 100);
  loadmatrix(ident_matrix);

   /*  draw earth background             */
  pushmatrix();
  lookat(0.0, 0.0, viewer_range, 0.0, 0.0, 0.0, 0.0);
  c3f(Color[EARTH]);
  circf(0.0,0.0,1.0);
  popmatrix();

  /* place our viewer                    */
  lookat(viewer_y, viewer_z, viewer_x, 0.0, 0.0, 0.0, 0.0);

  /*  draw the orbital ellipses          */
  for(sat = current_sat; sat != NULL; sat = sat->next) draw_orbit(sat);

  /*  draw the IJK axes                  */
  draw_axes();

  /*  draw the earth w/  rotation        */
  draw_globe(ROTATION_RATE * clock);

  /*  draw the satellites                */
  for(sat = current_sat; sat != NULL; sat = sat->next) draw_satellite(sat);

  swapbuffers();        /*  show it      */
}
```

```
/*  DRAW-SAT-VIEW                                              */
/*  Show that portion of the globe that is visible             */
/*      to the satellite.  This is best represented            */
/*      by a "fish-eye" view given by using perspective. */

draw_sat_view()
{
  float del_z;

  /* select the proper window and clear it               */
  winset(sat_view_gid);
  c3f(Color[BLACK]);
  zclear();
  clear();

  /*  if we're above the surface draw it */
  if (current_sat != NULL && current_sat->range > 1.001)
  {
    /* compute the fov based on the range and make the  */
    /* box deep enough to handle the near (viewable)    */
    /* half of the globe.                               */
    del_z = 1/current_sat->range;
    perspective(asin(del_z)*3790/PI, 1.0, (current_sat->range -1)/2,current_sat->range);
    loadmatrix(ident_matrix);

    pushmatrix();
    lookat(0.0, 0.0, current_sat->range, 0.0, 0.0, 0.0, 0.0);
    translate(0.0, 0.0, del_z);

    /*  draw a ring around the globe the color of the current satellite  */
    c3f(Color[current_sat->color]);
    circf(0.0, 0.0, 1.02 * sqrt(1 - del_z*del_z));

    /*  draw earth background    */
    c3f(Color[EARTH]);
    circf(0.0, 0.0, sqrt(1 - del_z*del_z));
    popmatrix();


    /* look from the satellite to the center of the earth      */
    /*  remember that IJK system not same as IRIS' XYZ system  */
    lookat(current_sat->y, current_sat->z, current_sat->x, 0.0, 0.0, 0.0, 0.0);

    /*  draw the earth w/ provided rotation  */
    draw_globe(ROTATION_RATE * clock);
  }
  swapbuffers();   /*  show it     */
}
```

```
/*  DRAW-MERC-VIEW                                         */
/*  Draw a mercator projection map and plot all            */
/*      lat/long                                           */

draw_merc_view()
{
  satellite     *sat;
  plot_struct   *plotter;
  int           k;              /* dummy loop counter */

  /*  get the right window              */
  winset(merc_view_gid);

  /* clear out any viewing or projection matrices left   */
  /*  on the window and set up a simple 2D screen        */
  ortho2(-DEG180-0.001,DEG180+0.001,-DEG120,DEG120);
  loadmatrix(ident_matrix);

  c3f(Color[BLACK]);     /*  go to black and clear window */
  clear();
  zclear();

  draw_mercator();       /*  draw the map                 */

  draw_sun();            /*  draw golden circle for sun   */

  for(sat = current_sat; sat != NULL; sat = sat->next) if (sat->plot)
  {
    c3f(Color[sat->color]);
    for(plotter = sat->plots; plotter != NULL; plotter = plotter->next)
      draw_dot(plotter->lon, plotter->lat);
  }
  swapbuffers();    /*  show it    */
}
```

```
/*    DRAW-DOT                                                */
/*    Used in drawing the Mercator map.                       */
/*    Draw a dot at the given x,y coordinate. Since it        */
/*    is a small dot, it is faster to draw an 8 point         */
/*    circle than the standard circf circle.                  */

draw_dot(x,y)
float  x,y;       /* coordinates of center        */
{
  move(x, y, 1.0);
  rpmv(-0.00873,-0.02107,0.0);
  rpdr(0.01234,0.01234,0.0);
  rpdr(0.0,0.01745,0.0);
  rpdr(-0.01234,0.01234,0.0);
  rpdr(-0.01745,0.0,0.0);
  rpdr(-0.01234,-0.01234,0.0);
  rpdr(0.0,-0.01745,0.0);
  rpdr(0.01234,-0.01234,0.0);
  pclos();
}




/*    DRAW-SUN                                                */
/*    Used in drawing the Mercator map.                       */
/*    Draw a transparent ellipse to represent the sun's       */
/*    position.                                               */

draw_sun()
{
  float sun_lat = SUN_INCL * sin(clock/SUN_TU),
        sun_lon = fmod(clock * (1/SUN_TU - ROTATION_RATE) - DEG180, DEG360);

  c4f(Color[SUN]);
  zbuffer(FALSE);
  blendfunction(BF_SA, BF_MSA);
  circf(sun_lon, sun_lat,1.0);          /* western hemisphere   */
  circf(sun_lon + DEG360, sun_lat, 1.0);  /* eastern hemisphere   */
  blendfunction(BF_ONE, BF_ZERO);
  zbuffer(TRUE);
}
```

```
/*  DRAW-PARAMETERS                                                   */
/*  Go through the appropriate parameter block and print              */
/*  each parameter in the right spot in the parameter window.         */

draw_parameters()
{
  ParBlock *sat_block;  /* current satellite parameter block  */
  int    k;             /* dummy loop counter  */

  winset(paramwin.gid);  /*  go to the parameter window    */
  c3f(Color[BLACK]);
  clear();

  ortho2(0,paramwin.sizex,0,paramwin.sizey);
  loadmatrix(ident_matrix);


  /*  draw satellite parameters                                   */

  c3f(Color[WHITE]);                        /*  write a title */
  cmov2i(10, paramwin.sizey - 3 * LINE);
  fmprstr("Satellite Parameters:");

  if (current_sat != NULL)
  {
    switch (param_option) {
      case CLASSICAL:   sat_block = &classicalparamblock; break;
      case SECONDARY:   sat_block = &secondaryparamblock; break;
      case POSITIONS:   sat_block = &positionparamblock; break;
      case OPTIONS:     sat_block = &optionsparamblock; break;
      case COMBINED:    sat_block = &combinedparamblock; break;
    }
    for (k = 5; k < sat_block->numparams + 5; k++)
    {
      cmov2i(10, paramwin.sizey - k * LINE);
      draw_parameter(sat_block->params[k - 5],current_sat->color,
                                       (int)current_sat-(int)&header_sat);
    }
  }


  /*  draw system parameters                                      */

  c3f(Color[WHITE]);                        /*  write a title */
  cmov2i(10, paramwin.sizey - 32 *LINE);
  fmprstr("System Parameters:");
  for (k = 34; k < sysparamblock.numparams + 34; k++)
  {
    cmov2i(10, paramwin.sizey - k * LINE);
    draw_parameter(sysparamblock.params[k - 34],WHITE,0);
  }
  swapbuffers();              /*   show it    */
}
```

```
/*  DRAW-PARAMETER                                          */
/*  Draw the contents of a parameter record in the                  */
/*     desired color.  Offset is used to get the value from         */
/*     the desired satellite.                                       */

draw_parameter(param,color_code,offset)
Par param;              /*   a parameter record        */
int color_code,         /*  what color to draw the data */
    offset;             /* offset of the string pointer */
{
  c3f(Color[WHITE]);
  fmprstr(param.name);
  param.value = (char *)((int)param.value + offset);

  c3f(Color[color_code]);
  switch (param.type) {         /* choose the correct output routine  */
  case DISTANCE:  draw_distance(param.value); break;
  case SPEED:     draw_speed(param.value); break;
  case FLOAT:     draw_float(param.value); break;
  case DEGREE:    draw_degree(param.value); break;
  case LAT:       draw_lat(param.value); break;
  case LON:       draw_lon(param.value); break;
  case TIME:      draw_time(param.value); break;
  case STRING:    fmprstr(param.value); break;
  case COLORCODE: fmprstr(color_names[*(int *)param.value]); break;
  case UNITCODE:  fmprstr(unit_names[*(int *)param.value]); break;
  case FLAG:      if(*(int *)param.value) fmprstr("On"); else fmprstr("Off"); break;
  }
}


/*  DRAW-xxxxxxx                                                    */
/*  Customized formatting routines that coerce the value            */
/*     into the format we want.                                     */

draw_distance(val)
char *val;
{
  char buf[15];

  sprintf(buf, "%4.2f %s", *(float *)val*distance_factor[unit_type],
                           distance_unit[unit_type]);
  fmprstr(buf);
}



draw_speed(val)
char *val;
{
  char buf[15];

  sprintf(buf, "%4.2f %s", *(float *)val*speed_factor[unit_type],
                           speed_unit[unit_type]);
  fmprstr(buf);
}
```

```
draw_float(val)
char *val;
{
  char buf[10];

  sprintf(buf, "%3.2f", *(float *)val);
  fmprstr(buf);
}


draw_degree(val)
char *val;
{
  char buf[10];

  sprintf(buf, "%3.2f", *(float *)val*TO_DEG);
  fmprstr(buf);
}


draw_lat(val)
char *val;
{
  char buf[10];

  if (*(float *)val >= 0.0)
    sprintf(buf, "%3.2f N", fabs(*(float *)val)*TO_DEG);
  else
    sprintf(buf, "%3.2f S", fabs(*(float *)val)*TO_DEG);
  fmprstr(buf);
}


draw_lon(val)
char *val;
{
  char buf[10];

  if (*(float *)val >= 0.0)
    sprintf(buf, "%3.2f E", fabs(*(float *)val)*TO_DEG);
  else
    sprintf(buf, "%3.2f W", fabs(*(float *)val)*TO_DEG);
  fmprstr(buf);
}


draw_time(val)
char *val;
{
  char buf[10];
  int   hour = *(float *)val / 60,
        min  = *(float *)val - hour * 60,
        sec  = (*(float *)val - (hour * 60 + min)) * 60;

  if (*(float *)val >= 0 || *(float *)val <= -60)
        sprintf(buf, "%02d:%02d:%02d", hour, abs(min), abs(sec));
  else  sprintf(buf, "-0:%02d:%02d", abs(min), abs(sec));
  fmprstr(buf);
}
```

```
/*  DRAW-SAT-LIST                                   */
/*  List out the linked list of satellites          */
/*  vertically in the satlistwin.                   */

draw_sat_list()
{
  satellite *sat;
  int    y = satlistwin.sizey - 3 * LINE;  /* first line of text */

  winset(satlistwin.gid);
  c3f(Color[BLACK]);              /*  clear the window */
  clear();

  ortho2(0,satlistwin.sizex,0,satlistwin.sizey);
  loadmatrix(ident_matrix);

  cmov2i(20, y);
  c3f(Color[WHITE]);              /*  write a title        */
  fmprstr("SMET Satellite List:");
  y = y - LINE;                   /*  blank line           */

  for(sat = current_sat; sat != NULL; sat = sat->next)
  {
    y = y - LINE;
    cmov2i(25,y);
    c3f(Color[sat->color]);       /*  current sat color    */
    fmprstr(sat->name);           /*  write a name         */
  }
  swapbuffers();                  /*  show it              */
}
```

```
/*  DRAW_INSTRUCTIONS                                              */
/*  Loop through the instructions array and draw it on the          */
/*      Main window, then wait for the user to press and           */
/*      release the rightmouse button.                             */

draw_instructions()
{
  int k;  /*  dummy loop variable   */

  winset(mainwin.gid);     /*  get the main window        */
  c3f(Color[BLACK]);
  clear();
  zclear();
  c3f(Color[CYAN]);

  /* clear out any viewing or projection matrices left  */
  /*  on the main window and set up a simple 2D screen  */
  ortho2(0,mainwin.sizex,0,mainwin.sizey);
  loadmatrix(ident_matrix);

  for (k = 0; k < 28; k++)
    {
      cmov2(75,700 - k * LINE);    /* move to the left margin    */
      fmprstr(instructions[k]);              /* draw the line of text      */
    }
  swapbuffers();          /*   show it    */

  while (qread(&k) != RIGHTMOUSE);  /*  wait for press  */
  while (qread(&k) != RIGHTMOUSE);  /* wait for release */
}
```

```
#/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#/*                                                                   */
#/*      makefile for SMET                                            */
#/*      by Carlos I. Noriega                                         */
#/*                                                                   */
#/*      Written - September 3, 1990                                  */
#/*      Modified - September 3, 1990                                 */
#/*                                                                   */
#/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
# compiler switches
CC       =      cc
INCLUDE  =      -I.
CFLAGS   =      -O -c -D$(LANGUAGE)
LIBS     =      -lmalloc -lfm -lgl_s -lm \
                imagesupport/libnpsimage.a ~zyda/utah/lib/librle.a -limage
SRCS     =      smet_addelete.c\
                smet_axes.c\
                smet_dialog.c\
                smet_files.c\
                smet_globe.c\
                smet_init.
                smet_kepler.c\
                smet_launch.c\
                smet_main.c\
                smet_maneuver.c\
                smet_map.c\
                smet_orbit.c\
                smet_param.c\
                smet_queue.c\
                smet_sat.c\
                smet_user.c\
                smet_windows.c\
                smet_vars.c
OBJS     =      smet_addelete.o\
                smet_axes.o\
                smet_dialog.o\
                smet_files.o\
                smet_globe.o\
                smet_init.o\
                smet_kepler.o\
                smet_launch.o\
                smet_main.o\
                smet_maneuver.o\
                smet_map.o\
                smet_orbit.o\
                smet_param.o\
                smet_queue.o\
                smet_sat.o\
                smet_user.o\
                smet_windows.o\
                smet_vars.o
smet:    $(OBJS)
         cc  -O $(INCLUDE) $(OBJS) -o smet $(LIBS)

clean:
         rm -f $(ALL)
         @echo "make clean new completed"

neat:
         rm -f .[BC]* .em* *.o *.BA* *.CK* core
         @echo "make neat new completed"

.c.o:
         $(CC) $(CFLAGS) $(INCLUDE) $<
```

LIST OF REFERENCES

1. Barrow, T. H., Yurchak, J. D., and Zyda, M. J., *Distributed Computer Communications in Support of Real-Time Visual Simulations*, Naval Postgraduate School, 1988.

2. Bate, R. R., Mueller, D. D., and White, J. E., *Fundamentals of Astrodynamics*, Dover Publications, Inc., 1971.

3. JSC-16970, *Indtroduction to Level-A Flight Design*, by E. L. Davis, Mission Planning and Analysis Division, NASA, Johnson Space Center, 1984.

BIBLIOGRAPHY

AU-15, *Space Handbook*, Air Command and Staff College, 1985.

Ball, K. J., and Osborne, G. F., *Space Vehicle Dynamics*, Oxford University Press, 1967.

Barrère, M., and others, *Rocket Propulsion*, Elsevier Publishing Company, 1960.

Blasingame, B. P., *Astronautics*, McGraw-Hill Book Company, 1964.

Buchanan, H. E., and Sperri, P., *Plane and Spherical Trigonometry*, Johnson Publishing Company, 1926.

Cornelisse, J. W., Schöyer, H. F. R., and Wakker, K. F., *Rocket Propulsion and Spaceflight Dynamics*, Pitman Publishing Limited, 1979.

Deutsch, R., *Orbital Dynamics of Space Vehicles*, Prentice-Hall, 1963.

Ehricke, K. A., *Space Flight, II. Dynamics*, D. Van Nostrand Company, Inc., 1962.

Penna, M. A., and Patterson, R. R., *Projective Geometry and its Application to Computer Graphics*, Prentice-Hall, 1986.

Reitz, H. L., Reilly, J. F., and Woods, R., *Plane and Spherical Trigonometry*, The Macmillan Company, 1936.

Silicon Graphics Inc., *4Sight User's Guide, Volume 1*, Mountain View, California, 1987.

Silicon Graphics Inc., *IRIS User's Guide*, Mountain View, California, 1987.

Silicon Graphics Inc., *IRIS User's Guide, GT Graphics Library User's Guide*, Mountain View, California, 1987.

Silicon Graphics Inc., *Parallel Programming on the IRIS-4D*, Mountain View, California, 1989.

Silicon Graphics Inc., *Tuning Graphics Code for Your IRIS-4D*, Mountain View, California, 1987.

Space Technology Laboratories, Inc., *Flight Performance Handbook for Powered Flight Operations*, John Wiley & Sons, Inc., 1962.

Space Trajectories, American Astronautical Society, Advanced Research Projects Agency and Radiation Incorporated, Academic Press, 1960.

Zyda, M. J., *Graphics and Video Laboratory, Books 1-8*, CS4202/CS4470 course notes, 1989-1990.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5100	2
3.	Commandant of the Marine Corps Code TE-06 Headquarters, U.S. Marine Corps Washington, D.C. 20360-0001	1
4.	Commander Naval Space Command Attn: Code N152 Dahlgren, VA 22448	1
5.	United States Space Command Attn: Technical Library Peterson AFB, CO 20350-2000	1
6.	Director Naval Space Systems Division (OP-943) Washington, D.C. 20350-2000	1
7.	Space Systems Academic Group, Code SP Naval Postgraduate School Monterey, CA 93943-5000	1
8.	Dr. Michael J. Zyda Code CS, Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	10

9. Dr. Herschell H. Loomis, Jr. 5
Code EC/LM
Naval Postgraduate School
Monterey, CA 93943-5000

10. Capt Carlos I. Noriega, USMC 3
6412 Redstone Circle
Colorado Springs, CO 80919

11. CPT Michael K. Weiderhold, USA 2
Headquarters, U.S. Space Command, Code J3SO
Peterson AFB, CO 80914-5603